FLEXERA SOFTWARE



FLEXERA SOFTWARE®
InstallShield®

InstallShield 2016

User Guide

Legal Information

Book Name: InstallShield 2016 User Guide

Part Number: ISP-2300-UG00

Product Release Date: August 2016

Copyright Notice

Copyright © 2016 Flexera Software LLC. All Rights Reserved.

This publication contains proprietary and confidential information and creative works owned by Flexera Software LLC and its licensors, if any. Any use, copying, publication, distribution, display, modification, or transmission of such publication in whole or in part in any form or by any means without the prior express written permission of Flexera Software LLC is strictly prohibited. Except where expressly provided by Flexera Software LLC in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera Software LLC intellectual property rights, whether by estoppel, implication, or otherwise.

All copies of the technology and related information, if allowed by Flexera Software LLC, must display this notice of copyright and ownership in full.

Intellectual Property

For a list of trademarks and patents that are owned by Flexera Software, see http://www.flexerasoftware.com/intellectual-property. All other brand and product names mentioned in Flexera Software products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.

Restricted Rights Legend

The Software is commercial computer software. If the user or licensee of the Software is an agency, department, or other entity of the United States Government, the use, duplication, reproduction, release, modification, disclosure, or transfer of the Software, or any related documentation of any kind, including technical data and manuals, is restricted by a license agreement or by the terms of this Agreement in accordance with Federal Acquisition Regulation 12.212 for civilian purposes and Defense Federal Acquisition Regulation Supplement 227.7202 for military purposes. The Software was developed fully at private expense. All other use is prohibited.

Contents

1 InstallShield 2016 Help Library 65

What's New in InstallShield 2016 67

What Was New in Earlier Versions of InstallShield 84

What's New in InstallShield 2015 SP1 84
What's New in InstallShield 2015 86
What's New in InstallShield 2014 SP1 100
What's New in InstallShield 2014 100
What's New in InstallShield 2013 SP1 112
What's New in InstallShield 2013 118
What's New in InstallShield 2012 Spring SP1 130
What's New in InstallShield 2012 Spring 131
What's New in InstallShield 2012 SP1 146
What's New in InstallShield 2012 146
What's New in InstallShield 2011 153
What's New in InstallShield 2010 Expansion Pack for Visual Studio 2010 172
What's New in InstallShield 2010 SP1 174
What's New in InstallShield 2010 177
What's New in InstallShield 2009 SP2 195
What's New in InstallShield 2009 SP1 197
What's New in InstallShield 2009 197
What's New in InstallShield 2008 213
What's New in InstallShield 12 SP1 233
What's New in InstallShield 12 234

Target System Requirements 239

Targeting 64-Bit Operating Systems 240
Targeting 64-Bit Operating Systems with Basic MSI and InstallScript MSI Installations 241
Targeting 64-Bit Operating Systems with InstallScript Installations 244
Targeting 64-Bit Operating Systems with Suite/Advanced UI and Advanced UI Installations 245

Launching InstallShield with vs. Without Administrative Privileges 246

Developing and Building Installations on 32-Bit vs. 64-Bit Systems 247
Using Help 249
Contacting Us 251

2 Getting Started 253
Installation Fundamentals 254
Application Lifecycle 255
Starting InstallShield 256
InstallShield Start Page 256
Working with Projects 257
Installation Projects Overview 258
Determining Which Installation Project Is Right for You 258
Project Types 259
Advanced UI and Suite/Advanced UI Projects 261
Basic MSI Installation Projects 262
Run-Time Behavior for Basic MSI Installations with InstallScript Custom Actions 262
DIM Projects 263
InstallScript Installation Projects 263
InstallScript MSI Installation Projects 264
Run-Time Behavior for InstallScript MSI Installations 265
Merge Module Projects 266
MSI Database and MSM Database Projects 267
InstallScript Object Projects 267
QuickPatch Projects 267
Transform Projects 268
Using Projects 268
Creating New Projects 268
Opening Projects 269
Opening Projects from Source Code Control 269
Opening Patch Creation Properties Files in Direct Edit Mode 269
Opening Windows Installer Packages 270
Opening Merge Modules 270
Opening Object Projects 270
Placing Merge Modules in the Redistributables View 271
Saving Projects 271
Saving a Project with a New Name and Location 271
Converting from One Project Type to Another Project Type 272
Converting a Basic MSI Project to an InstallScript MSI Project 272
Converting an InstallScript MSI Project to an InstallScript Project 273
GUIDs 273
Changing the Default Project Location 275
Reusing Project Elements 275
Using a Repository to Share Project Elements 276
Setting up a Network Repository 276
Project Templates 276

Creating Project Templates *277*
Basing New Projects on Templates *277*
Publishing Project Templates to a Repository *278*
Sample Projects 278
Project Assistant 279
Using the Project Assistant 279
Using the More Options, Other Places, and Help Links Sections in the Project Assistant 280
Navigating in the Project Assistant 280
Opening the Installation Designer 281
Showing or Hiding the Project Assistant 281
Application Information Page 282
Add or Remove Programs in the Control Panel 282
Company Name and Product Name in Your Installation 282
Installation Requirements Page 283
Specifying Operating System Requirements in the Project Assistant 283
When Does the Installation Check for Requirements? 284
Modifying the Run-Time Message for Software Requirements 284
Creating Custom Installation Requirements 284
Installation Architecture Page 285
Adding Features in the Project Assistant 285
Determining Whether to Create a Multiple-Feature Installation 286
Creating Installations with Multiple Features 286
Default Features 287
Defining Feature Hierarchy 287
Application Files Page 288
Adding Files to Features in the Project Assistant 288
Removing Files from Features in the Project Assistant 289
Adding Files to a Fixed Folder Location 289
Viewing Additional Predefined Folders 289
Application Shortcuts Page 290
File Extensions 290
Creating Shortcuts to Files That Are Not Included in the Installation 291
Modifying a Default Shortcut in the Project Assistant 291
Associating a Shortcut's Target with a File Extension in the Project Assistant 292
Application Registry Page 292
Updating the Registry 293
Configuring Registry Data in the Project Assistant 293
Modifying Registry Data Values in the Project Assistant 294
Associating Registry Data with Features 294
Using Variable Data Types in Registry Data 294
Application Paths 295
Installation Interview Page 295
Specifying Dialogs for Your Installation in the Project Assistant 296
Allowing End Users to Modify the Installation Location 296
License Agreements 297
Creating Selectively Installable Installations 297

Installation Localization Page. . . . 298
Localizing String Data from the Project Assistant . . . 298
How Localized String Data Is Used in the Installation. . . . 299
Build Installation Page. . . . 299
Building Your Installation from the Project Assistant . . . 299
After Completing the Project Assistant: Next Steps . . . 301
Working with the InstallShield Interface . . . 301
Displaying the View List . . . 301
Opening Views in the InstallShield User Interface . . . 302
Working with the Group Box Area in Various Views . . . 302
Showing or Hiding Toolbars . . . 304
Adding Buttons and Menus to a Toolbar . . . 304
Removing Buttons and Menus from a Toolbar. . . . 305
Creating Custom Toolbars . . . 305
Docking or Undocking the Output Window . . . 305
Working with the Script Editor Pane in Various Views . . . 306
Using Bookmarks in the Script Editors. . . . 306
Enabling or Disabling Auto Completion in the Script Editors . . . 307
Using Auto Completion when Writing Code in the Script Editors . . . 308
Enabling or Disabling InstallScript Function Call Tips in the Script Editor. . . . 309
Viewing Function Call Tips for an InstallScript Function in the Script Editor. . . . 309
Enabling or Disabling Syntax Highlighting in the Script Editors . . . 310
Changing Colors for Syntax Highlighting in the Script Editors . . . 311
Changing the Font that Is Used to Display Text in the Script Editors. . . . 312
Showing or Hiding Line Numbering in the Script Editors . . . 312
Going to a Line Number in a Script that Is Displayed in a Script Editor. . . . 313
Enabling or Disabling Automatic Indentation in the Script Editors . . . 313
Setting the Tab Width for the Script Editors. . . . 313
Enabling or Disabling Syntax Folding in the Script Editors . . . 314
Showing or Hiding Whitespace in the Script Editors . . . 315
Dragging and Dropping Text in the Script Editors . . . 315
Printing a Script File from Within a Script Editor . . . 316
Keyboard Shortcuts for the Script Editors. . . . 316
Configuring Advanced Settings for InstallShield. . . . 317
Changing the Timestamp Server for Digital Signatures . . . 318
Configuring the Compression Level for Files that Are Streamed into Setup.exe and ISSetup.dll . . . 319
Configuring the Maximum Size for .cab Files . . . 321
Modifying the List of Portable Executable Files for the Standalone Build . . . 323
Adding Support for XML Encoding Options . . . 324
Specifying the Location Where All Virtual Packages Should Be Built . . . 325
Sharing Virtual Machine Settings for Distribution of Releases . . . 326
Upgrading from Earlier InstallShield Versions . . . 327
Upgrading Projects from InstallShield 2014 or Earlier. . . . 328
Upgrading Projects from InstallShield 2013 or Earlier. . . . 331
Upgrading Projects from InstallShield 2012 Spring or Earlier . . . 331
Upgrading Projects from InstallShield 2012 or Earlier. . . . 334

Upgrading Projects from InstallShield 2011 or Earlier . . . 336
Upgrading Projects from InstallShield 2010 or Earlier . . . 339
Upgrading Projects from InstallShield 2009 or Earlier . . . 343
Upgrading Projects from InstallShield 2008 or Earlier . . . 353
Upgrading Projects from InstallShield 12 or Earlier . . . 363
Upgrading Projects from InstallShield 11.5 or Earlier . . . 366
Upgrading InstallShield 11.5 or Earlier Basic MSI Projects that Have InstallScript Custom Actions . . . 368
Creating and Scheduling InstallScript Custom Actions that Call InstallScript Event Handlers for Basic MSI Projects371
Upgrading InstallShield 11.5 or Earlier InstallScript MSI Projects . . . 373
Creating and Scheduling InstallScript Custom Actions that Call InstallScript Event Handlers for InstallScript MSI Projects 377
Upgrading InstallShield 11.5 or Earlier InstallScript Projects . . . 381
Upgrading InstallShield 11.5 or Earlier QuickPatch Projects that Have InstallScript Custom Actions . . . 383
Creating Standard Patches for InstallShield 11.5 and Earlier InstallScript MSI Projects . . . 383
Upgrading InstallShield 11.5 or Earlier InstallScript MSI Object Projects or Projects that Contain This Type of Object383
Upgrading Projects Created with InstallShield Express . . . 384
Upgrading Projects Created with InstallShield Professional . . . 384
Migrating from InstallShield Professional 6.x . . . 385
Migrating from InstallShield Professional 5.x . . . 388
Script Changes: Lexicon Conversion . . . 394
Adding Project Assistant Dialog Support to Projects Upgraded from InstallShield Professional . . . 397
Upgrading Projects Created with InstallShield—Windows Installer Edition . . . 398
Post-Upgrade Changes in the Property Manager . . . 399
Upgrading from Other InstallShield Editions . . . 399
Upgrading from the Express Edition to the Professional or Premier Editions . . . 404
Upgrading from the Professional Edition to the Premier Edition . . . 404
Converting or Importing Visual Studio Projects into InstallShield Projects . . . 404
Obtaining Updates for InstallShield . . . 408
Run-Time Language Support in InstallShield . . . 408
Code Page Requirements for Language Support . . . 409
Installing Supplemental Language Support on a Build Machine . . . 410
Supported Application Programming Languages . . . 410

3 Tutorials . . . 411

InstallScript Project Tutorial . . . 413
Step 1: Creating, Building, and Testing Projects . . . 413
Creating a Project with the Project Assistant . . . 414
Specifying Application Information . . . 415
Customizing the Installation Architecture . . . 415
Adding Files to Your Project . . . 415
Creating Shortcuts . . . 416
Configuring Registry Data . . . 416
Selecting Dialogs with the Installation Interview . . . 416
Choosing a Language for Your Installation . . . 417
Building Your Installation . . . 417

Running Your Installation *417*
Working with the InstallShield Interface 418
Setting Feature Properties *418*
Setting Setup Type Properties *418*
Creating Components and File Links *419*
Building a Release 419
Naming the Release *420*
Selecting the Media Type and General Options *420*
Specifying a Password and Supported Platforms *420*
Specifying Setup Languages and Including Features *421*
Defining Media Layout and Dialog Appearance *421*
Specifying Internet Options and Digitally Signing Your Application *421*
Specifying Update and Postbuild Information *422*
Reviewing Your Settings *422*
Troubleshooting Your Installation 422
Step 2: Shortcuts and Registry Data 422
Creating Shortcuts 423
Creating Registry Data 424
Step 3: Registering COM Servers 425
Step 4: Conditions and Properties 426
Step 5: Working with Scripts 426
Step 6: Changing the User Interface 429
Handling User Input 429
Changing the Dialogs Displayed 429
Using the Dialog Editor 430
Basic MSI Project Tutorial 431
Step 1: Creating, Building, and Testing Your Project 431
Creating a New Basic MSI Project 432
Specifying Application Information *433*
Setting Installation Requirements *434*
Customizing Installation Architecture *434*
Adding Files to Your Project *435*
Creating Shortcuts *435*
Configuring Registry Data *436*
Selecting Dialogs with the Installation Interview *436*
Choosing a Language for Your Installation *436*
Building Your Installation *437*
Running Your Installation *437*
Working in the IDE 438
Setting Feature Properties *438*
Creating Components and File Links *439*
Building a Release 439
Naming the Product Configuration and Release *440*
Specifying Filtering Settings and Languages *440*
Selecting the Media Type and Disk Spanning Options *441*
Specifying Compression Settings and Setup Launcher Options *441*

Adding Digital Signature and Password Protection *442*
Including .NET Framework Support and Choosing Advanced Settings *442*
Reviewing Your Settings *443*
Troubleshooting Your Installation 443
Step 2: Shortcuts and Registry Data 444
Creating Shortcuts 444
Creating Registry Data *445*
Step 3: Registering COM Servers 446
Step 4: Conditions and Properties 448
Step 5: Changing the End-User Interface 449
Adding a New Dialog 449
Modifying Dialog Layout in the Dialog Editor 450
Globalization Tutorial 451
Opening the Project File 451
Selecting the Target Languages 451
Editing Language-Specific String Entries 452
Creating String Entries 453
Including Language-Specific Files and Components 454
Specifying Component Installation Conditions 454
Translating the Strings 455
Building the Installation 456
Running the Installation 457
Testing the Installation 457

4 Creating Installations 459

Before You Begin 460
Requirements for the Windows Logo Program 461
Introduction to Windows Installer 461
Minimizing the Number of User Account Control Prompts During Installation 462
Setting an Application's Disk Usage 468
INSTALLDIR vs. TARGETDIR 468
Preventing the Current Installation from Overwriting a Future Major Version of the Same Product 468
Preparing Installations for Non-Administrator Patches 470
Settings for Platforms and Platform Suites 471
Specifying Installation Information 475
Configuring General Project Settings 475
Saving an InstallShield Project File (.ism) in XML or Binary Format 475
Specifying a Product Name 475
Specifying the Product Version 476
Product Version Numbers in InstallScript and InstallScript Object Projects 477
Setting the Product Code in a Windows Installer–Based Project 478
Setting the Product Code in an InstallScript-Based Project 478
Setting the Upgrade Code 479
Setting Product Conditions 479
Setting the Default Product Destination Folder (INSTALLDIR) 481
Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment 482

Selecting the Locked-Down Permissions Type for a Project *485*
Specifying Whether Windows Installer Installations Should Be Logged 486
Running an InstallScript Installation Multiple Times 487
Configuring the Enable Maintenance Setting 488
Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations 488
Specifying the InstallScript User Interface Type for InstallScript MSI Installations *494*
Entering Summary Information Stream Data 494
Accessing the Summary Information Stream Panel 495
Setting Summary Information Stream Properties 495
Using the Template Summary Property 495
Configuring Add or Remove Programs Information 496
Specifying a Readme File 497
Including a Software Identification Tag for Your Product 498
Including Microsoft System Center Configuration Manager Application Model Data About Your Product 499
Organizing Files for Your Installation 501
Designing Installations 501
Separating Applications into Components 502
Separating Applications into Features 502
Using Path Variables 502
Types of Path Variables 503
Predefined Path Variables *504*
Standard Path Variables *505*
Registry Path Variables *506*
Environment Path Variables *507*
Creating and Defining a Custom Path Variable 508
Overriding the Value of a Custom Path Variable for a Release 509
Converting Absolute Path References in Your Project to Path Variables 510
Deleting a Path Variable 511
Including Files and Folders 511
Adding to Your Project Files to Be Installed on Target Systems 511
Dragging and Dropping Files Using the Context Menu 513
Adding Files from Your 64-Bit Source Machine's 64-Bit System32 Folder 514
Creating New Destination Folders 515
Creating Empty Folders 515
Specifying Hard-Coded Destination Directories 515
Removing Files and Folders from Target Systems 516
Tips for Managing Files and Folders in Your Project 518
Finding Files and Folders in Your Project 519
Displaying Components in the Files and Folders View 519
Modifying Components through the Files and Folders View 519
Configuring Settings for a File 520
File Associations 520
Adding New File Extensions *521*
Removing File Extensions *521*
Setting File Extension Properties *522*
Specifying Command Verbs 522

Removing a Verb from a File Extension . . . 522
Adding New MIME Types . . . 523
Removing MIME Types . . . 523
Creating ProgIDs . . . 523
Removing ProgIDs . . . 524
Dynamic File Linking . . . 525
Limitations of Dynamic File Linking . . . *526*
Determining the Appropriate Component Creation Method for Dynamically Linked Files . . . *527*
Creating a Dynamic Link . . . *528*
Adding Dynamic File Links to Components . . . *529*
Setting a Key File for a Dynamic File Link . . . *530*
Overwriting Files and Components on the Target System . . . 531
Displaying Predefined Folders in the Files View . . . 532
Companion Files . . . 532
Configuring Permissions for Files and Folders . . . 533
Extracting COM Data When Files Are Added . . . 534
Installing Self-Registering Files . . . 535
Identifying Properties and Dependencies of .NET Assemblies . . . 535
Scanning 64-Bit .NET Assemblies for Dependencies . . . *537*
Reviewing .NET Dependency Scanner Results . . . *538*
Installing Fonts Through InstallScript and InstallScript Object Projects . . . 539
Installing Fonts to the Windows Fonts Folder . . . *540*
Enabling or Disabling Global Font Registration . . . *540*
Enabling or Disabling Registration of a Font File . . . *540*
Using Components . . . 541
Setup Best Practices . . . 541
Component Creation . . . 543
Using the Setup Design or Components Views to Create a Component . . . *543*
Using the Best Practices Option with the Component Wizard . . . *544*
Best Practice Rules for Creating Components . . . 545
Letting InstallShield Create a COM Server Component . . . 546
Letting InstallShield Create Font Components . . . 546
Using the Component Type Option with the Component Wizard . . . *547*
Installing COM Servers . . . 548
Installing Fonts . . . 548
Exporting Components to Other Projects . . . *548*
Deleting a Component in Your Project . . . *549*
Component-Feature Associations . . . 550
Associating New Components with Features . . . *550*
Associating Existing Components with Features . . . *551*
Disassociating Components from Features . . . *551*
Adding Data to a Component . . . 552
Adding Files to Components . . . *552*
Changing the Value in the Link To Column for a Component's File . . . *553*
Deleting a File from a Component . . . *553*
Adding Registry Data to Components . . . *554*
Creating Shortcuts in the Components View . . . *554*

Adding Subfolders to Statically Linked Components . . . 555
Component Key Files . . . 555
Setting Component Key Files . . . 555
Clearing a Key File from a Component . . . 556
Component Settings . . . 556
Component Destination vs. Feature Destination . . . 557
Setting a Destination Folder for the Component's Files . . . 558
Specifying a Component's Destination from the Script . . . 559
Uppercase Directory Identifiers and Component Destinations . . . 560
Specifying Whether a Component's Files and Other Associated Data Are Uninstalled During Uninstallation . . . 561
Configuring Component Conditions . . . 562
Reevaluating Component Conditions During Reinstallation . . . 563
Managing Reference Counts for Shared Files . . . 563
Checking File Versions Before Installing . . . 565
Installing Files of the Same Name . . . 565
Setting a Component's Remote Installation Setting . . . 566
Component's Remote Installation Setting vs. Feature's Remote Installation Setting . . . 567
Extracting COM Registration Data at Build Time . . . 567
Scanning for .NET Dependencies and Properties . . . 568
Specifying the .NET Application File . . . 569
Reading Properties Passed to the .NET Installer Class . . . 570
Enabling and Disabling Registry Reflection . . . 571
Specifying Whether Shared Component Patching Should Be Enabled for a Component . . . 572
Advanced Component Settings . . . 575
Configuring COM Registration Settings Manually . . . 575
Configuring COM Classes for COM Registration Manually . . . 576
Configuring ProgIDs for COM Registration Manually . . . 577
Configuring Type Libraries for COM Registration Manually . . . 578
Registering a File Extension . . . 579
Installing Assemblies . . . 580
Adding Assemblies . . . 580
Deleting Assemblies . . . 581
Testing for .NET Assembly Support on the Target System . . . 581
Specifying an Application Path for a Component . . . 582
Configuring Device Driver Settings . . . 583
Publishing Components . . . 583
Specifying Publishing Information . . . 584
Adding ComponentIDs for a Component to Be Published . . . 585
Removing ComponentIDs . . . 585
Adding Qualifiers to ComponentIDs . . . 586
Removing Qualifiers from ComponentIDs . . . 586
Configuring Qualifier Settings . . . 587
Defining Features . . . 587
Creating Features . . . 588
Configuring Feature Settings . . . 588
Setting a Feature's Destination . . . 589
Setting Feature Conditions . . . 590

Displaying Features to End Users . . . 592
Conditionally Selecting Features . . . 592
Conditionally Hiding Features . . . 593
Requiring Features to Be Installed . . . 594
Advertising Features . . . 595
Configuring a Feature's Install Level Setting . . . 596
Setting a Feature's Remote Installation Setting . . . 596
Using Release Flags with Features . . . 597
Assigning Release Flags to Features . . . *597*
Removing Release Flags . . . *598*
Reordering Features . . . 598
Using the Required Features Setting . . . 599
Working with Setup Types . . . 599
Adding Setup Types . . . 600
Editing Setup Types . . . 600
Renaming Setup Types . . . 601
Deleting Setup Types . . . 601
Including Redistributables in Your Installation . . . 602
Shipping Redistributable Files . . . 603
Managing the Redistributables Gallery . . . 605
Downloading Redistributables to Your Computer . . . *607*
Adding InstallShield Prerequisites to the Redistributables Gallery . . . *609*
Removing InstallShield Prerequisites from the Redistributables Gallery . . . *609*
Browsing for Merge Modules . . . *610*
What Happens When You Browse for a Merge Module . . . *610*
Adding Merge Modules to the Redistributables Gallery . . . *611*
Removing Merge Modules from the Redistributables Gallery . . . *611*
Registering Objects in InstallScript Projects . . . *612*
Incorporating InstallShield Prerequisites, Merge Modules, and Objects in Projects . . . 612
Adding InstallShield Prerequisites, Merge Modules, and Objects to Basic MSI and InstallScript MSI Projects . . . *612*
Adding InstallShield Prerequisites, Merge Modules, and Objects to InstallScript Projects . . . *614*
Removing InstallShield Prerequisites from a Project . . . *615*
Removing Merge Modules and Objects from a Project . . . *615*
Determining the Files in InstallShield Prerequisites, Merge Modules, and Objects . . . *616*
Working with InstallShield Prerequisites that Are Included in Installation Projects . . . 616
Setup Prerequisites vs. Feature Prerequisites . . . *617*
Associating an InstallShield Prerequisite with a Feature in a Basic MSI Project . . . *619*
Disassociating an InstallShield Prerequisite from a Feature in a Basic MSI Project . . . *620*
Specifying the Installation Order of InstallShield Prerequisites . . . *620*
Configuring a Release that Includes InstallShield Prerequisites . . . *621*
Specifying the Directories that Contain InstallShield Prerequisites . . . *622*
Specifying a Run-Time Location for a Specific InstallShield Prerequisite . . . *623*
Assigning Release Flags to InstallShield Prerequisites . . . *624*
Building a Release that Includes InstallShield Prerequisites . . . *625*
Run-Time Behavior for an Installation that Includes InstallShield Prerequisites . . . *625*
Uninstalling an Application Whose Installation Included InstallShield Prerequisites . . . *629*

Working with Merge Modules that Are Included in Installation Projects . . . 630
Specifying the Directories that Contain Merge Modules . . . *630*
Overriding a Merge Module's Destination . . . *631*
Troubleshooting Merge Module Issues . . . *632*
Publishing a Merge Module to a Repository from Within an Installation Project . . . *632*
Adding Windows Installer Redistributables to Projects . . . 633
Including Microsoft Windows Installer Prerequisites . . . *634*
Adding .NET Framework Redistributables to Projects . . . 635
Including the Microsoft .NET Framework and Microsoft .NET Framework Language Pack Prerequisites . . . *637*
Including the MySQL Connector ODBC Prerequisite . . . 638
Including the InstallShield Prerequisite for the Oracle 11g Instant Client . . . 639
Including the DirectX 9.0 Object . . . 639
Identifying Application Dependencies . . . 640
Static Scanning . . . 641
Dynamic Scanning . . . 642
Scanning for 64-Bit Dependencies . . . 642
Reviewing Dependency Scanner Results . . . 643
Filtering Files in Dependency Scanners . . . 643
Registering COM Servers . . . 645
Traditional COM Registration . . . 646
Determining Whether a COM Server Supports Self-Registration . . . *646*
Extracting COM Information from a COM Server . . . *647*
Filtering Registry Changes for COM Extraction . . . *648*
Self-Registering COM Servers . . . *650*
Self-Registration Methods . . . 651
InstallShield Self-Registration (ISSelfReg) . . . 652
Registry-Free COM Registration . . . 652
Creating and Modifying Reg-Free COM Files for Your Installation . . . *653*
Sample Manifest File for a Reg-Free COM File . . . *654*
Including DIMs in a Project . . . 654
Determining the Appropriate Method for Incorporating DIMs in Installation Projects . . . 655
Referencing a DIM in a Project . . . 655
Associating a DIM Reference with a Feature . . . *656*
Resolving Build-Time Conflicts Between a Basic MSI Project and a DIM Reference . . . *657*
Viewing Build Instructions for a DIM Reference . . . *658*
Overriding the Destination for a DIM Reference . . . *658*
Scheduling Custom Actions and Dialogs from a DIM Reference . . . *659*
Opening a Referenced DIM Project from Within an Installation Project . . . *659*
Importing a DIM into a Project . . . 660
Resolving Design-Time Conflicts While Importing a DIM Into a Basic MSI Project . . . *660*
Identifying DIM Elements in an Installation Project . . . 661
Overriding Path Variables in a DIM Project for Use in an Installation Project . . . 661
Configuring the Target System . . . 663
Creating Shortcuts and Program Folders . . . 663
Types of Shortcuts . . . 663
Creating Shortcuts . . . 665

Configuring Shortcut Settings . . . 666
Specifying the Icon for a Shortcut . . . 667
Creating Internet Shortcuts . . . 668
Creating Shortcuts to Folders . . . 668
Specifying a Keyboard Shortcut for Accessing a Shortcut . . . 669
Renaming Shortcuts . . . 670
Setting Shell Properties for a Shortcut . . . 671
Suppressing Initial Pinning of a Shortcut to the Windows 8 Start Screen . . . 671
Specifying Whether End Users Should Be Able to Pin a Shortcut to the Taskbar or Start Menu . . . 673
Preventing a Shortcut on the Start Menu from Being Highlighted as Newly Installed . . . 675
Setting Custom Shell Properties . . . 676
Creating Uninstallation Shortcuts for Basic MSI Projects . . . 677
Creating Uninstallation Shortcuts for InstallScript and InstallScript MSI Projects . . . 678
Configuring the Appearance of a Desktop App's Tile on the Start Screen . . . 679
Editing the Registry . . . 680
Filtering Registry Entries by Component or Feature . . . 681
Refreshing the Registry View . . . 682
Creating Registry Keys . . . 683
Dragging and Dropping Registry Entries to Create Registry Keys . . . 684
Removing Registry Keys . . . 685
Creating Registry Values . . . 686
Modifying Registry Value Data . . . 687
Removing Registry Values . . . 688
Registry Flags . . . 688
Searching for Registry Entries in a Project . . . 690
Setting Uninstall Behavior for Registry Keys . . . 691
Using Environment Variables in Registry Values . . . 692
Writing Property Values to the Registry . . . 692
Importing Registry Files . . . 693
Exporting Registry Files . . . 693
Setting Key Paths for Components . . . 694
Configuring Permissions for Registry Keys . . . 695
Primary Keys for the Registry Table . . . 695
Specifying a Primary Key for the Registry Table . . . 696
Entering Multi-Line String Values for Registry Keys . . . 696
Writing Entries Under HKCU for a Per-User Installation . . . 697
Setting or Getting Multi-Line Strings in the Registry . . . 698
Working with Registry Functions . . . 699
Reading Data from the Registry . . . 699
Changing .ini File Data . . . 700
Creating .ini File References . . . 701
Importing an Existing .ini File . . . 702
Specifying a Section in an .ini File . . . 702
Specifying a Keyword in an .ini File . . . 703
Reading Data from .ini Files . . . 704
Reading All Key Names from .ini Files . . . 704

Configuring ODBC Resources . . . 705
Installing ODBC Resources . . . 705
Including Additional ODBC Resources . . . 706
Associating ODBC Resources with Features . . . 706
Setting ODBC Resource Attributes . . . 707
Using Environment Variables . . . 708
Setting Environment Variables . . . 708
Environment Variables Example . . . 709
Modifying XML Files . . . 710
Overview of XML File Changes . . . 711
Run-Time Requirements for XML File Changes . . . 712
Creating an XML File Reference . . . 713
Specifying the Location of the XML File . . . 714
Associating an XML File Change Reference with a Feature . . . 715
Adding a Root Element . . . 715
Adding a New Element . . . 716
Adding an Element for a .NET Configuration File . . . 717
Adding an Attribute to an Element . . . 717
Editing an Attribute's Value . . . 718
Adding Content to an Element . . . 719
Using XPath Expressions to Find XML Data in an XML File . . . 719
Using Windows Installer Properties to Dynamically Modify XML Files . . . 724
Using InstallScript Text Substitution to Dynamically Modify XML Files . . . 725
Using Reserved Characters (<, >, &, ', and ") Inside Elements . . . 726
Using Namespaces in XML Files . . . 727
Declaring Namespace Mappings for an XML File . . . *728*
Adding a Namespace Prefix to an Element . . . *728*
Adding a Namespace Prefix to an Attribute . . . *729*
Removing a Namespace Prefix from an Attribute . . . *730*
Removing a Namespace Prefix from an Element . . . *730*
Removing Namespace Mappings from an XML File . . . *731*
Testing Installation Changes to an XML File . . . 731
Testing Uninstallation Changes to an XML File . . . 733
Removing an Element or an XML File from the XML File Changes View . . . 734
Modifying Text Files . . . 734
Creating a Text Change Set . . . 735
Specifying a Change for a Text File . . . 736
Changing the Order in Which Text File Changes Are Made . . . 737
Using Windows Installer Properties to Dynamically Modify Text Files . . . 737
Specifying the Code Page that Should Be Used for Opening ANSI Text Files . . . 740
Removing a Change Item or a Change Set from the Text File Changes View . . . 740
Scheduling Tasks . . . 741
Adding a Scheduled Task . . . 741
Using Windows Installer Properties to Dynamically Configure a Scheduled Task . . . 742
Removing a Scheduled Task . . . 743
Installing, Controlling, and Configuring Windows Services . . . 744

Creating Predetermined User Accounts at Run Time . . . 745
Per-User vs. Per-Machine Installations . . . 747
Running InstallScript Installations Without Administrative Privileges . . . 749
Customizing Installation Behavior . . . 751
Using InstallScript . . . 751
Overview of ISSetup.dll . . . 752
Script Files . . . 752
Creating InstallScript Files . . . *753*
Opening an InstallScript File . . . *753*
Inserting and Importing Script Files . . . *754*
Compiling Scripts . . . *755*
Debugging Scripts . . . *756*
Using Preprocessor Statements to Debug the Script . . . 756
Renaming an InstallScript File . . . *757*
Using String Entries in Scripts . . . *757*
Removing InstallScript Files from Projects . . . *757*
Creating Script Libraries (.obl Files) . . . *758*
Publishing InstallScript Files (.rul and .h) to a Repository . . . *759*
InstallScript Lists . . . *759*
Creating and Destroying Lists . . . 760
Adding Elements to Lists . . . 761
Adding an Element to an Empty List . . . 762
Adding an Element Before the Current Element . . . 762
Adding an Element After the Current Element . . . 763
Adding Elements Before and After the Current Element . . . 763
Changing Existing Elements in a List . . . 764
Deleting Elements from a List . . . 765
Finding a Particular Element in a List . . . 766
Getting the First and Next Elements in a List . . . 766
Reading a File into a List . . . 767
Setting an Index in a List . . . 767
Traversing Lists . . . 768
Writing a List to a File . . . 769
Saving InstallScript Files . . . *769*
System Restore . . . *769*
Getting and Setting Properties . . . *770*
Using Bit Flags . . . *771*
String Comparisons . . . *772*
Using Null-Delimited Strings . . . *773*
Relative Path . . . *773*
Long File Names . . . *774*
Long File Names and 16-bit Applications . . . 774
Long File Names and Double Quotation Marks . . . 774
Long File Name Format . . . 774
Defining Constants Through the Compiler . . . *775*
Using Windows Constants in a Script . . . *775*
Coding Long String Literals . . . *775*

Absolute Path . . . 775
Building Functions . . . 775
Calling Functions . . . 776
Calling a DLL File Function . . . 777
Passing an Array to a DLL File Function . . . 778
Declaring Functions . . . 778
Returning Values from Functions . . . 779
Unsupported Functions . . . 780
Writing Entry-Point Functions . . . 781
Adding Function Calls with the Function Wizard . . . 782
Declared Windows API Functions . . . 782
Specifying a Non-Default Feature Event Handler Function . . . 782
Accessing Global Variables . . . 783
Uninstalling Initialization (.ini) File Entries . . . 784
General Limitations of Uninstalling Initialization (.ini) File Entries . . . 784
Uninstalling AddProfString's Initialization (.ini) File Entries . . . 784
Uninstalling ReplaceProfString's Initialization (.ini) File Entries . . . 785
Uninstalling WriteProfString's Initialization (.ini) File Entries . . . 786
Extending Your Installation with COM Objects . . . 786
Specifying Features and Subfeatures in Function Calls . . . 789
Calling a Windows API Function . . . 789
Embedding Custom Transfer File Operations . . . 790
Expressing Large Numbers in InstallScript . . . 792
Installing Device Drivers . . . 793
Checking the Compiler Version . . . 793
Checking the Authoring Environment with _ISCRIPT_ISDEV and _ISCRIPT_ISPRO . . . 794
Launching an Installation from Another InstallScript Installation . . . 794
Language Support for InstallScript . . . 795
Preventing Color Distortion . . . 796
Using Custom Actions . . . 798
Using the Custom Action Wizard . . . 798
Creating Custom Actions in the Custom Actions and Sequences View (or the Custom Actions View) . . . 799
Cloning Custom Actions . . . 800
Exporting Custom Actions to Other Projects . . . 801
Configuring Custom Action Settings . . . 801
Custom Action Type Overview . . . 802
Custom Action Return Values . . . 803
In-Script Execution . . . 803
Documenting the Behavior of Custom Actions . . . 805
Guidelines for Creating Custom Actions that Meet Requirements of the Windows Logo Program . . . 806
Conditionally Launching Custom Actions Based on Release Flags . . . 807
Placing Files in the .msi Database and Extracting Them During Run Time . . . 807
Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions . . . 809
InstallScript Custom Actions . . . 810
Calling Functions in Standard DLL Files . . . 811
Calling Functions in Windows Installer DLL Files . . . 812

Passing Parameters to a DLL File Function in a Custom Action . . . 814
Calling a Public Method in a Managed Assembly . . . 815
Run-Time Requirements for Managed-Code Custom Actions . . . *815*
Specifying the Signature for a Managed Method in an Assembly Custom Action . . . *816*
Using 32-Bit vs. 64-Bit Managed-Code Custom Actions . . . *818*
Calling a Kill-Process Custom Action . . . 819
Calling a PowerShell Custom Action . . . 820
Launching Executable Files . . . 822
Using Msiexec.exe to Launch a Second Windows Installer Installation . . . 824
Nested Installations . . . 826
Creating Nested Installation Custom Actions . . . *826*
Inserting Nested Installation Custom Actions into Sequences . . . *827*
Removing Child Products . . . *828*
Calling MessageBoxA in an Installation . . . 829
Searching for Files on the Target System . . . 831
Launching the Application After the Installation Is Complete . . . 833
Exiting the Installation from Within an InstallScript Custom Action . . . 833
Changing ODBC Properties Through Script . . . 833
Using the INSTALLDIR Property in a VBScript Custom Action . . . 834
Using Action Text . . . 834
Specifying an Action Description and a Template for Action Data . . . 835
Displaying Action Descriptions on the Progress Dialog . . . 835
Displaying Action Data on the Progress Dialog . . . 836
Removing an Action Description or a Template for Action Data . . . 838
Defining Sequences . . . 838
Installation Sequence . . . 839
Advertisement Sequence . . . 850
Administration Sequence . . . 852
User Interface Sequence . . . 854
Execute Sequence . . . 854
Inserting Actions into Sequences . . . 855
Copying a Custom Action from One Sequence to Another . . . 856
Reordering a Sequence . . . 857
Removing Actions from Sequences . . . 858
Sequencing Rollback Custom Actions . . . 858
Sequencing a Custom Action that Launches an .exe File . . . 859
Sequencing a Custom Action that Calls a Function in a DLL File . . . 860
Sequencing a Custom Action that Calls a Script . . . 861
Sequencing Custom Actions that Set Properties or Directory Properties . . . 863
Sequencing a Custom Action that Launches a Second .msi Package . . . 864
ISSetAllUsers Custom Action . . . 864
Using Support Files . . . 865
Adding Support Files . . . 866
Adding a License File . . . 866
Sorting Support Files . . . 867
Adding Files and Folders to the Disk1 Folder . . . 867

Removing Files or Folders from the Disk1 Folder . . . 868
Adding Files and Folders to the Last Disk Folder . . . 868
Removing Files or Folders from the Last Disk Folder . . . 869
Adding Files and Folders to the Other Disk Folder . . . 869
Removing Files or Folders from the Other Disk Folder . . . 870
Removing Support Files . . . 870
Defining the End-User Interface . . . 871
Working with Dialogs . . . 871
Working with Dialogs in Any Project Type . . . 871
Using the Dialog Wizard to Create a New Dialog . . . 871
Adding the Ability to Create or Set an Existing User Account . . . 872
Exporting a Dialog to an .isd File for Use in Other Projects . . . 874
Importing Dialogs from an .isd File . . . 874
Publishing a Dialog (.isd) File to a Repository . . . 875
Exporting All Dialogs to an .rc File . . . 875
Importing Dialogs from Resource .dll Files . . . 876
Exporting Dialogs to Other Projects . . . 877
Removing Dialogs from Projects . . . 878
Creating and Configuring Custom Dialogs . . . 878
Undoing Changes in the Dialog Editor . . . 879
Working with Dialogs in InstallScript and InstallScript MSI Projects . . . 879
Displaying Dialogs During InstallScript and InstallScript MSI Installations . . . 880
Changing the Text on Dialogs in InstallScript and InstallScript MSI Projects . . . 881
Background About the Default Images on Dialogs in InstallScript and InstallScript MSI Projects . . . 881
Creating New Custom Dialogs in InstallScript and InstallScript MSI Projects . . . 882
Using the New Dialog Wizard to Add a New Custom Dialog to an InstallScript or InstallScript MSI Project . . . 882
Editing the Layout of a Dialog in an InstallScript or InstallScript MSI Project . . . 883
Adding a Control to a Dialog in an InstallScript or InstallScript MSI Project . . . 884
Setting a Control's Properties . . . 884
Displaying Controls on Top of a Bitmap . . . 885
Using InstallScript to Implement Custom Dialogs . . . 885
Using InstallScript to Process Dialog Controls . . . 888
Editing Dialog Behavior in an InstallScript or InstallScript MSI Project . . . 891
Adding a Field That Contains a Product Name, Product Version, or Installed Version in Sd Dialog Static Text Fields . 891
Using an HTML Control on a Dialog . . . 892
Reverting to the Default Dialog . . . 894
Resource Compiler and Resource Linker . . . 894
Dialog Sampler . . . 895
Dialog Skins . . . 895
Specifying Dialog Skins . . . 896
Working with Dialogs in Basic MSI Projects . . . 897
Displaying Dialogs During Basic MSI Installations . . . 898
Creating New Dialogs in Basic MSI Projects . . . 899
Editing Dialog Layout in Basic MSI Projects . . . 899
Opening the Dialog Editor . . . 900
Adding a Control to a Dialog in a Basic MSI Project . . . 900

Configuring the Controls' Settings . . . 900
Displaying Controls on Top of a Bitmap . . . 901
Editing Dialog Behavior in Basic MSI Projects . . . *901*
Using Control Events in Basic MSI Dialogs . . . *902*
Using Subscriptions in Basic MSI Dialogs . . . *905*
Triggering Control Events in Basic MSI Dialogs . . . *907*
Adding New Events to a Dialog Control . . . 907
Reordering a Dialog Control's Events . . . 907
Removing an Event from a Dialog Control . . . 908
Launching Custom Actions from Dialog Buttons . . . *908*
Viewing End-User Dialog Order in the Custom Actions and Sequences view (Basic MSI Projects) . . . *908*
CustomSetup Dialog Options . . . *910*
Dialog Themes . . . *912*
Run-Time Requirements for the Wide Dialog Themes . . . 912
Previewing a Dialog Theme . . . 912
Selecting or Changing a Dialog Theme . . . 913
Applying a Theme to a Custom Exterior Dialog . . . 914
Adding a Logo or Other Image to the Exterior Dialogs . . . 914
Available Themes and Corresponding Dialog Sizes . . . 915
Circles Theme (Wide) . . . 916
Classic Theme . . . 917
Cooperation Theme (Wide) . . . 918
Filmstrip Theme (Wide) . . . 919
Global Theme . . . 920
InstallShield Blue Theme . . . 921
InstallShield Blue Theme (Wide) . . . 922
InstallShield Silver Theme . . . 923
Monitor Theme . . . 924
Pastel Wheat Theme . . . 925
Theater Theme (Wide) . . . 926
Dialog Support for Right-to-Left Languages . . . *927*
Launching a File Open Dialog . . . *928*
Requiring End Users to Scroll Through the EULA in the LicenseAgreement Dialog . . . *931*
Adding a Print Button to a Dialog . . . *933*
Minimizing Reboots on Windows Vista and Later Systems . . . *935*
Dialog Controls . . . 935
Billboard Control . . . *939*
Bitmap Control . . . *941*
Check Box Control . . . *943*
Combo Box Control . . . *948*
Dialog Control . . . *952*
Directory Combo Control . . . *955*
Directory List Control . . . *958*
Edit Field Control . . . *960*
Group Box Control . . . *964*
Hyperlink Control . . . *967*
Icon Control . . . *970*
Line Control . . . *973*

List Box Control . . . 975
List View Control . . . 979
Masked Edit Control . . . 984
Path Edit Control . . . 986
Progress Bar Control . . . 989
Push Button Control . . . 992
Radio Button Control . . . 996
Radio Button Group Control . . . 998
Scrollable Text Control . . . 1002
Selection Tree Control . . . 1004
Text Area Control . . . 1008
Volume Cost List Control . . . 1012
Volume Select Combo Control . . . 1015
Copying and Pasting Controls . . . 1018
Cutting and Pasting Controls . . . 1018
Working with the Wizard Interface . . . 1019
Elements of the Wizard Interface . . . 1019
Using Styles, Brushes, and Control Themes to Customize the Wizard Interface . . . 1021
Defining a Custom Style for the Wizard Interface . . . 1022
Applying a Text Style to Text in the Wizard Interface . . . 1023
Defining a Custom Brush for the Wizard Interface . . . 1024
Specifying the Background Brush for the Header, Body, and Navigation Areas of the Wizard Interface . . . 1025
Overriding the Default Background Brushes for a Wizard Page or Secondary Window . . . 1026
Defining a Custom Control Theme for the Wizard Interface . . . 1027
Applying a Theme to a Control in the Wizard Interface . . . 1027
Selecting the Format for the Wizard Interface . . . 1029
Creating a New Blank Wizard Page in an Advanced UI or Suite/Advanced UI Project . . . 1030
Adding a Predefined Wizard Page in an Advanced UI or Suite/Advanced UI Project . . . 1030
Creating a New Blank Secondary Window in an Advanced UI or Suite/Advanced UI Project . . . 1031
Sequencing Wizard Pages . . . 1032
Editing the Layout and Behavior of a Wizard Page or Secondary Window . . . 1032
Adding a Control to a Wizard Page or Secondary Window . . . 1033
Changing the Tab Order of Controls on a Wizard Page or Secondary Window . . . 1034
Designating Keyboard Shortcuts and Using Ampersands (&) in the Wizard Interface . . . 1034
Using Navigation Buttons on Wizard Pages . . . 1036
Overriding the Default Behavior of a Navigation Button on a Wizard Page . . . 1037
Populating Combo Box and List Box Controls in the Wizard Interface . . . 1038
Configuring Validation for a Control on a Wizard Page or Window . . . 1040
Configuring an Action for an Element in the Wizard Interface . . . 1042
Using Images, Text Files, and Other Objects in the Wizard Interface . . . 1048
DPI-Awareness in the Wizard Interface . . . 1048
Localizing the End-User Interface . . . 1049
Working with String Entries in Projects that Support Multiple Languages . . . 1050
Using String Entries in InstallShield . . . 1051
Adding a String Entry . . . 1053
Editing a String Entry . . . 1054

Searching for Instances of String Identifiers . . . 1055
Finding and Replacing String Entries . . . 1056
Removing a String Entry . . . 1057
Removing String Identifiers from Localizable Settings . . . 1057
Displaying Billboards . . . 1058
Displaying Billboards in Basic MSI Installations . . . 1058
Billboard File Types for Basic MSI Projects . . . 1058
Types of Billboards for Basic MSI Projects . . . 1059
Specifying Which Type of Billboard to Use in a Basic MSI Project . . . 1062
Adding an Adobe Flash Application File Billboard to a Basic MSI Project . . . 1062
Adding Image Billboards to a Basic MSI Project . . . 1063
Configuring Billboard Settings in a Basic MSI Project . . . 1063
Previewing Billboards Without Building and Launching a Release . . . 1064
Setting the Billboard Order in a Basic MSI Project . . . 1064
Run-Time Behavior of a Basic MSI Installation that Includes Billboards . . . 1064
Removing a Billboard from a Basic MSI Project . . . 1065
Displaying Billboards in InstallScript and InstallScript MSI Installations . . . 1066
Billboard Styles and File Types for InstallScript and InstallScript MSI Projects . . . 1066
Naming Billboard Files in an InstallScript or InstallScript MSI Project . . . 1068
Adding an Adobe Flash Application File Billboard to an InstallScript or InstallScript MSI Project . . . 1069
Adding an Image Billboard to an InstallScript or InstallScript MSI Project . . . 1070
Adding or Modifying the Code in an InstallScript or InstallScript MSI Project to Display Billboards . . . 1071
Setting the Billboard Order in an InstallScript or InstallScript MSI Project . . . 1071
Displaying Billboards with Special Effects During an InstallScript or InstallScript MSI Installation . . . 1072
Moving Billboards to a Different Screen Location for an InstallScript or InstallScript MSI Project . . . 1072
Removing a Billboard from an InstallScript or InstallScript MSI Project . . . 1072
Displaying a Background Window in InstallScript and InstallScript MSI Installations . . . 1073
Populating List Boxes at Run Time . . . 1074
Displaying File Browse Dialogs . . . 1074
Displaying Network Browse Dialogs in InstallScript Installations . . . 1074
Preparing Installations for Update Notifications . . . 1077
Enabling Automatic Update Notifications for a Project . . . 1077
Disabling Automatic Update Notifications for a Project . . . 1078
Files that Need to Be Installed for Automatic Update Notification . . . 1079
Creating a Shortcut to Check for Updates . . . 1079
Adding a Check-for-Updates Check Box to the SetupComplete Dialog . . . 1080
Registering Your Application with FlexNet Connect . . . 1081
Configuring Servers . . . 1083
Configuring SQL Support . . . 1083
Using Windows Installer Properties for SQL Login Settings . . . 1084
Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties . . . 1085
Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects . . . 1086
Adding a New SQL Connection . . . 1087
Overriding the Default TCP/IP Network Library with a Different Protocol for a SQL Server Database . . . 1088
Requirements for Connecting to Instances of SQL Server Express LocalDB . . . 1088
Adding a New SQL Script . . . 1089

Inserting and Importing SQL Scripts . . . 1090
Importing a SQL Server Database and Generating a SQL Script File . . . 1091
Editing a SQL Script File . . . 1092
Specifying a Version Number for a SQL Script File . . . 1093
Specifying the Order for Running Multiple SQL Scripts That Are Associated with a Connection . . . 1094
Overriding the Default SQL Run-Time Behavior . . . 1095
Handling SQL Run-Time Errors . . . 1096
Setting Up a Database Server Type Condition for SQL Scripts . . . 1097
Conditionally Launching a SQL Script in a Windows Installer-Based Installation . . . 1098
Conditionally Launching a SQL Script in an InstallScript Installation . . . 1099
Requiring that SQL Scripts Be Run Only Against a Full SQL Server . . . 1100
Using Windows Installer Properties to Dynamically Replace Strings in SQL Scripts . . . 1100
Using InstallScript Text Substitution to Dynamically Replace Strings in SQL Scripts . . . 1102
Requiring a SQL Server-Side Installation for a Windows Installer-Based Project . . . 1104
Requiring a Server-Side Installation for an InstallScript Project . . . 1105
Publishing SQL Script Files to a Repository . . . 1105
Installing the MySQL ODBC Driver . . . 1106
Deleting a SQL Server Database During Uninstallation . . . 1106
Connecting to a Microsoft Windows Azure Database Server and Running SQL Scripts . . . 1107
Connecting to an Instance of Oracle and Running SQL Scripts . . . 1108
Downloading the Oracle Instant Client and Creating an .msi Package for It . . . 1109
Downloading the Oracle Basic Instant Client and the Oracle ODBC Instant Client and Creating an .msi Package and InstallShield Prerequisite for Both . . . 1110
Installing the Oracle 11g Instant Client . . . 1114
Running SQL Scripts with Unicode Characters on an Oracle Database Server . . . 1115
Creating a Sample Installation that Creates a SQL Server Database by Running Customized SQL Script . . . 1116
Creating a Sample Installation that Creates an Oracle Schema by Running Customized SQL Script . . . 1118
Adding a SQLLogin Predefined Wizard Page in a Suite/Advanced UI Project . . . 1120
Managing COM+ Applications and Components . . . 1121
Managing COM+ Server Applications . . . 1122
Managing COM+ Application Proxies . . . 1123
Including a COM+ Application that Targets Servers and Client Machines . . . 1124
Managing Internet Information Services . . . 1125
Version-Specific Information for IIS Support in InstallShield . . . 1125
Determining Which Version of IIS Is Installed . . . 1126
Run-Time Requirements for IIS Support . . . 1127
Specifying Whether a Web Server Should Allow the CMD Command to Be Used for SSI #exec Directives . . . 1128
Creating a Web Site and Adding an Application or a Virtual Directory . . . 1129
Scanning an IIS Web Site and Importing Its Settings into an InstallShield Project . . . 1130
Filtering IIS Data When Importing a Web Site and Its Settings into an InstallShield Project . . . 1131
Creating a Nested Virtual Directory . . . 1133
Configuring the TCP Port and Site Numbers . . . 1134
Specifying the IIS Host Header Name for a Web Site . . . 1137
Specifying the SSL Certificate for a Web Site . . . 1138
Adding Files to an IIS Virtual Directory . . . 1138
Removing Applications and Virtual Directories from the Internet Information Services View . . . 1139
Adding Application Pools . . . 1140

Removing Application Pools from the Internet Information Services View . . . 1140
Adding Web Service Extensions . . . 1141
Removing Web Service Extensions from the Internet Information Services View . . . 1142
Feature and Component Associations for IIS Support . . . 1142
Uninstalling Web Sites, Applications, and Virtual Directories . . . 1143
Uninstalling Web Service Extensions and Application Pools . . . 1144
Setting the ASP.NET Version for a Web Site or Application . . . 1145
Considerations for Supporting IIS 6 on 64-Bit Platforms . . . 1146
Defining Application Mappings for a Web Site, Application, or Virtual Directory . . . 1148
Specifying Timeout Parameters for a Web Site, Application, or Virtual Directory . . . 1148
Determining If a Target System Has IIS 6 or Earlier or the IIS 6 Metabase Compatibility Feature . . . 1149
Configuring Advanced IIS Settings . . . 1150
Configuring Custom Error Messages for a Web Site, Application, or Virtual Directory . . . 1151
Using Windows Installer Properties to Dynamically Modify IIS Settings . . . 1152
Using InstallScript Text Substitution to Dynamically Modify IIS Settings . . . 1154
Deploying Web Services on a Target Machine . . . 1155
Adding IISROOTFOLDER Support . . . 1155
IIS_WEBSITE_NAME Property . . . 1156
IIS_PORT_NUMBER Property . . . 1156

Preparing Installations for Maintenance and Uninstallation . . . 1157
Preparing an InstallScript Installation for Uninstallation . . . 1157
Executing Script Code Only During Uninstallation or Only During Installation . . . 1158
InstallScript Functions that Are Logged for Uninstallation . . . 1159
Maintenance/Uninstallation Feature . . . 1160
Removing Files that Were Created by Your Product . . . 1160
Removing Registry Data Created by Your Product . . . 1161

Configuring Multiple Packages for Installation Using Transaction Processing . . . 1163
Overview of Multiple-Package Installations that Use Transaction Processing . . . 1163
Adding a New Chained .msi Package to Your Project . . . 1164
Assigning Release Flags to a Chained .msi Package . . . 1165
Removing a Chained .msi Package from Your Project . . . 1166

Building, Testing, and Deploying Installations . . . 1167
Creating and Building Releases . . . 1167
Working with Product Configurations . . . 1167
Selecting the Appropriate Type of Architecture Validation for Builds . . . 1168
Working with Releases . . . 1171
Using the Release Wizard to Create and Build a Release . . . 1171
Using the Releases View to Create and Build a Release . . . 1172
Setting the Release Location . . . 1173
Building a Release from the Command Line . . . 1173
Using ISCmdBld.exe to Build a Release from the Command Line . . . 1173
Passing Command-Line Build Parameters in an .ini File . . . 1174
Using ISBuild.exe to Build from the Command Line . . . 1178
Building a Self-Extracting Executable File from the Command Line . . . 1179
Rebuilding Releases . . . 1179
Performing Quick Builds . . . 1180

Performing Batch Builds *1180*
Specifying Commands that Run Before, During, and After Builds *1181*
Resolving Build Errors and Warnings *1183*
Changing Project Settings from the Command Line 1183
Canceling Builds *1184*
Standalone Build *1184*
Installing the Standalone Build on a Build Machine 1184
Adding Redistributables to the Build Machine for the Standalone Build 1186
Standalone Command-Line Build 1188
Standalone Automation Interface 1188
Installing the Standalone Build and InstallShield on the Same Machine 1189
Microsoft Build Engine (MSBuild) *1189*
Using MSBuild to Build a Release from the Command Line 1196
Configuring Release Settings 1196
Creating a Single Executable File for Distribution *1197*
Leaving Files Uncompressed in a Disk Image *1198*
Specifying the Required Execution Level for Your Setup Launcher on Windows Vista and Later Platforms *1199*
Specifying Whether a Product Should Be Advertised If Its InstallShield Prerequisites Are Run with Elevated Privileges *1201*
Password-Protecting Installations *1202*
Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level *1203*
Specifying the Run-Time Location for Advanced UI or Suite/Advanced UI Packages at the Release Level *1204*
Digital Signing and Security 1205
Digitally Signing a Release and Its Files at Build Time *1207*
Digitally Signing a Release After It Has Been Built From the Command Line *1208*
Release Flags 1208
Product Configuration Flags vs. Release Flags *1210*
Filtering Based on Release Flags *1211*
Using One Project to Create Installations for Multiple Product Configurations 1212
Typical Web Installation vs. One-Click Install 1213
One-Click Install Installations in InstallScript Projects *1214*
One-Click Install Objects *1215*
Player Object *1215*
LastError Method 1215
Open Method 1216
Ether Object *1217*
GetLastError Method 1217
IsPlaying Method 1219
Play Method 1219
SetProperty Method 1220
Digital Signatures for One-Click Install Installations *1220*
Replacing Obsolete Properties and Methods *1221*
Default Web Page (Setup.htm) *1222*
System Requirements for One-Click Install Installations *1223*
Alternate Ways to Call the Setup Player *1223*
HTTPS and One-Click Install Installations *1223*
Creating One-Click Install Installations *1223*

Using the Setup Player . . . 1224
Passing Data to the Launched One-Click Install Installation . . . 1225
Running an Internet Installation Silently . . . 1225
Setting the Run-Time Language . . . 1225
Running a One-Click Install Installation in Debug Mode . . . 1226
Authenticating a User from InstallScript . . . 1226
Setup.ini . . . 1226
The [Startup] Section . . . *1227*
The [Mif] Section . . . *1229*
Cloning Releases . . . 1230
Creating UWP App Packages . . . 1231
Testing and Running Installations . . . 1232
Test Running Installations . . . 1232
Running Installations from the InstallShield Interface . . . 1232
Running Installations in Silent Mode . . . 1233
InstallScript MSI and InstallScript Silent Installations . . . *1233*
Creating a Silent Installation . . . *1234*
Creating the Response File . . . *1235*
Response File Silent Header . . . 1236
Response File Application Header . . . 1237
Response File Dialog Sequence . . . 1237
Response File Dialog Data . . . 1238
Sample Response File . . . 1247
Running the Silent Installation . . . *1248*
Checking for Errors Using the Setup.log File . . . *1249*
MSI Silent Mode Installations for InstallScript MSI Projects . . . 1251
Silent Uninstallations from the Command Line (InstallScript MSI Only) . . . 1252
Validating Projects . . . 1252
Validating an Installation Package or Merge Module . . . 1253
Specifying Whether Validation Should Be Performed at Build Time . . . 1254
Specifying Which ICEs, ISICEs, ISVICEs and ISBPs Should Be Run During Validation . . . 1254
Validating an Installation Package or Merge Module on Demand . . . 1255
Viewing Validation Results . . . 1255
ICEs . . . 1256
ISICEs . . . 1256
ISICE01 . . . *1259*
ISICE02 . . . *1260*
ISICE03 . . . *1260*
ISICE04 . . . *1261*
ISICE05 . . . *1261*
ISICE06 . . . *1262*
ISICE07 . . . *1263*
ISICE08 . . . *1264*
ISICE09 . . . *1265*
ISICE10 . . . *1265*
ISICE11 . . . *1266*
ISICE12 . . . *1268*

ISICE16 1268
ISICE17 1269
ISICE18 1270
ISICE19 1270
ISICE20 1273
ISICE21 1274
ISICE22 1274
ISICE23 1275
ISICE24 1276
ISICE25 1276
ISICE26 1277
InstallShield Virtualization Suitability Suite 1278
ISVICE01 1280
ISVICE02 1281
ISVICE03 1282
ISVICE04 1282
ISVICE05 1283
ISVICE06 1284
ISVICE07 1284
ISVICE08 1285
ISVICE09 1286
ISVICE10 1286
ISVICE11 1287
ISVICE12 1288
ISVICE13 1289
ISVICE14 1289
ISVICE15 1290
ISVICE16 1291
ISVICE17 1292
ISVICE18 1292
ISVICE19 1293
InstallShield UWP App Suitability Suite 1293
ISUWP01 1295
ISUWP02 1296
ISUWP03 1296
ISUWP04 1296
ISUWP05 1297
ISUWP06 1297
ISUWP07 1297
ISUWP08 1298
ISUWP09 1298
ISUWP10 1299
ISUWP11 1299
ISUWP12 1300
ISUWP13 1300
ISUWP14 1301

ISUWP15 . . . . . *1301*
ISUWP16 . . . . . *1301*
ISUWP17 . . . . . *1302*
ISUWP18 . . . . . *1302*
ISUWP19 . . . . . *1302*
ISUWP20 . . . . . *1303*
InstallShield Best Practice Suite 1303
ISBP01 . . . . . *1305*
ISBP02 . . . . . *1306*
ISBP03 . . . . . *1306*
ISBP04 . . . . . *1307*
ISBP05 . . . . . *1308*
ISBP06 . . . . . *1308*
ISBP07 . . . . . *1309*
ISBP08 . . . . . *1310*
ISBP09 . . . . . *1311*
ISBP10 . . . . . *1311*
ISBP11 . . . . . *1312*
ISBP12 . . . . . *1313*
ISBP13 . . . . . *1313*
ISBP14 . . . . . *1314*
ISBP15 . . . . . *1315*
ISBP16 . . . . . *1316*
ISBP17 . . . . . *1316*
ISBP18 . . . . . *1317*
ISBP19 . . . . . *1317*
ISBP20 . . . . . *1318*
Understanding When an Installation or Uninstallation Restarts the Target System 1319
Debugging Windows Installer–Based Installations 1320
Starting the MSI Debugger 1321
Setting Breakpoints in the MSI Debugger 1321
Removing Breakpoints 1322
Viewing and Setting Properties in the MSI Debugger 1322
Step Into Actions in the MSI Debugger 1323
Step Through Actions in the MSI Debugger 1323
Stopping the MSI Debugger 1324
Spanning Installations over Multiple Disks or CDs 1324
Compressing Multi-Disk Installations 1324
Disk Spanning Rules 1325
Custom Disk Spanning 1325
Creating a Setup Launcher 1327
Customizing File Properties for the Setup Launcher 1329
Specifying the Icon for the Setup Launcher 1334
Distributing Installations 1335
Distributing Releases to a Folder or FTP Site Automatically 1335
Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand 1336

Preparing Installations for Internet Distribution . . . 1338
Redistributable Files that Are Distributed with an InstallScript Installation . . . 1339
Deploying UWP App Packages . . . 1339
Creating Transforms . . . 1341
Creating a Transform by Comparing Two .msi Packages . . . 1342
Creating a Transform Based on a Single .msi Package . . . 1342
Applying Transforms . . . 1343
Editing Transforms . . . 1343
Saving .msi Files as Transforms . . . 1344
Saving Transforms as .msi Files . . . 1344
Response Transforms . . . 1345
Creating a Response Transform . . . 1345
Modifying a Response Transform . . . 1345
Testing a Response Transform . . . 1346
Configuring Server Locations . . . 1346
Adding Server Locations . . . 1346
Modifying Server Locations . . . 1347
Removing Server Locations . . . 1347
Changing the Order of Server Locations . . . 1348

5 Creating Advanced UI and Suite/Advanced UI Installations . . . 1349

Advanced UI Projects vs. Suite/Advanced UI Projects . . . 1351
Organizing Features, Packages, Prerequisites, and Files in an Advanced UI or Suite/Advanced UI Installation . . . 1354
Guidelines for Adding Packages to an Advanced UI or Suite/Advanced UI Project . . . 1355
Adding an .msi Package, an .msp Patch, or a Transaction to an Advanced UI or Suite/Advanced UI Project . . . 1356
Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project . . . 1358
Special Considerations for InstallScript Packages in Advanced UI and Suite/Advanced UI Projects . . . 1359
Adding an InstallShield Project (.ism) as a Package in a Suite/Advanced UI Project . . . 1361
Adding an Executable Package (.exe) to a Suite/Advanced UI Project . . . 1362
Adding a Web Deploy Package to a Suite/Advanced UI Project . . . 1363
Adding a Sideloading UWP App Package (.appx) to a Suite/Advanced UI Project . . . 1364
Static vs. Dynamic Files for Packages in an Advanced UI or Suite/Advanced UI Project . . . 1365
Creating a Dynamic Link in an Advanced UI or Suite/Advanced UI Project . . . 1366
Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project . . . 1368
Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project . . . 1369
Primary Packages vs. Dependency Packages in Advanced UI and Suite/Advanced UI Projects . . . 1370
Specifying the Installation Order of Packages and Transactions in an Advanced UI or Suite/Advanced UI Project . . . 1371
Configuring Settings for a Package in an Advanced UI or Suite/Advanced UI Project . . . 1372
Associating a Package in an Advanced UI or Suite/Advanced UI Project with a Feature . . . 1373
Sharing Common Packages Among Different Advanced UI and Suite/Advanced UI Installations . . . 1373
Specifying a Run-Time Location for a Specific Package in an Advanced UI or Suite/Advanced UI Project . . . 1375
Configuring Package Operations for an Advanced UI or Suite/Advanced UI Installation . . . 1376
Using Custom Folder Names for Packages in Advanced UI and Suite/Advanced UI Installations . . . 1378
Customizing the Behavior of a Suite/Advanced UI Installation . . . 1378
Enabling Windows Roles and Features During a Suite/Advanced UI Installation . . . 1379
Using Actions to Extend the Behavior of a Suite/Advanced UI Installation . . . 1382

Working with an .exe File for an Action in a Suite/Advanced UI Installation 1383
Working with a DLL File for an Action in a Suite/Advanced UI Installation. 1384
Working with a PowerShell Script for an Action in a Suite/Advanced UI Installation 1387
Working with an Action that Sets a Property in a Suite/Advanced UI Installation. 1389
Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation 1390
Working with a Managed-Code Action in a Suite/Advanced UI Installation 1391
Adding an Action to a Suite/Advanced UI Project. 1395
Configuring a Suite/Advanced UI Action's Settings 1395
Scheduling a Suite/Advanced UI Action 1396
Types of Events in a Suite/Advanced UI Installation 1398

Defining Exit Conditions for an Advanced UI or Suite/Advanced UI Installation. 1399

Triggering Install Mode or Maintenance Mode of an Advanced UI or Suite/Advanced UI Installation 1401

Add or Remove Program Entries for an Advanced UI or Suite/Advanced UI Installation. 1402

Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation. 1403

Using Advanced UI and Suite/Advanced UI Formatted Expressions to Dynamically Configure Command Lines 1406

Defining the End-User Interface of an Advanced UI or Suite/Advanced UI Installation 1407

Referring to Feature States and Other Feature Data in the UI of an Advanced UI or Suite/Advanced UI Project .1408

Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time 1410

Writing Object Expressions in Advanced UI and Suite/Advanced UI Projects to Search Target Systems 1412

Minimizing the Number of User Account Control Prompts During Advanced UI and Suite/Advanced UI Installations 1413

Restarting a Target System for an Advanced UI or Suite/Advanced UI Package. 1415

Specifying the Behavior for an Advanced UI or Suite/Advanced UI Package that Requires a Restart. 1416

Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation 1418

Password-Protecting an Advanced UI or Suite/Advanced UI Installation. 1422

Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation 1423

Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation 1427

6 Designing InstallShield Prerequisites and Other Redistributables 1431

Defining InstallShield Prerequisites. 1433

Creating an InstallShield Prerequisite. 1433
Modifying an Existing InstallShield Prerequisite. 1434
Setting Properties for an InstallShield Prerequisite 1434
Specifying an Alternate URL for a .prq File 1434
Setting Installation Conditions for an InstallShield Prerequisite 1435
Adding a New Installation Condition for an InstallShield Prerequisite 1436
Adding an Operating System Condition for an InstallShield Prerequisite. 1436
Modifying an Installation Condition for an InstallShield Prerequisite 1437
Removing an Installation Condition from an InstallShield Prerequisite. 1438
Specifying Files for an InstallShield Prerequisite 1438
Specifying URLs for Downloading InstallShield Prerequisites 1439
Specifying Parameters for Installing an InstallShield Prerequisite. 1440
Specifying Command-Line Parameters for an InstallShield Prerequisite. 1441

Restarting a Target Machine for InstallShield Prerequisite Installations 1443
Specifying Installation Behavior for an InstallShield Prerequisite 1444
Specifying that an InstallShield Prerequisite Requires Administrative Privileges 1444
Allowing End Users to Choose Whether to Install an InstallShield Prerequisite 1445
Specifying Whether to Include the Name of a Prerequisite in the List of Setup Prerequisites to Be Installed on the Target System 1445
Specifying Whether to Show the Progress of an InstallShield Prerequisite Installation at Run Time 1447
Planning for Issues that Could Occur with an InstallShield Prerequisite Installation 1448
Specifying the Behavior for an InstallShield Prerequisite that Requires a Restart 1448
Specifying Dependencies for an InstallShield Prerequisite 1448
Adding a Dependency for an InstallShield Prerequisite 1449
Modifying a Dependency for an InstallShield Prerequisite 1449
Removing a Dependency from an InstallShield Prerequisite 1449
Saving an InstallShield Prerequisite 1449
Designing Merge Modules 1451
Creating a Merge Module Project 1451
Specifying the Default Destination Folder for a Merge Module 1452
Selecting the Merge Module Language 1452
Adding Dependencies to Merge Modules 1453
Adding Exclusions to Merge Modules 1454
Publishing a Merge Module to a Repository from Within a Merge Module Project 1455
Creating InstallScript Objects 1457
Designing Objects 1458
Designing Features for Objects 1459
Managing Object Files 1460
Designing an Object's Wizard 1461
Custom Object Wizards 1462
Localizing an Object's Design-Time Environment 1463
Preparing an Object for Internationalization 1463
Making Your Object Compatible with Installations During Run Time 1464
Properties and Methods 1465
Creating a Property 1465
Creating a Structure-Valued Property 1467
Passing a String Array Between an Object and an Installation 1469
Accessing an Object's Properties 1470
Creating a Method 1471
Using Methods 1471
Object Status Properties 1472
Creating the Object's Run-Time User Interface 1475
Unsupported Features in Objects 1476
Building an Object 1476
Building an Object from the User Interface 1477
Compiling and Building an Object from the Command Line 1478
Testing and Debugging an Object 1478
Distributing an Object 1479
Object Scripts 1480

Functions Not Supported in Object Projects . . . 1480
Constants Not Supported in Object Projects . . . 1480
Functions Supported Only in Object Projects . . . 1481
Adding a ScriptDefinedVar Property to Your Object . . . 1481
Using System Variables in Objects . . . 1482
Setting Your Object's Destination Through the Script . . . 1483

7 Modularizing Installation Projects to Distribute Development Work and Enable Reuse . . . 1485

Specifying Installation Information for a DIM . . . 1485
Saving a DIM Project File in XML or Binary Format . . . 1485
Specifying the Default Destination Folder for a DIM . . . 1486
Using Path Variables in a DIM . . . 1486
Organizing Files for a DIM . . . 1490
Configuring the Target System for a DIM . . . 1491
Customizing Installation Behavior for a DIM . . . 1491
Configuring Servers for a DIM . . . 1492

8 Updating Applications . . . 1493

Upgrades Overview . . . 1493
Major Upgrades . . . 1494
Minor Upgrades . . . 1495
Small Updates . . . 1495
Patching . . . 1496
QuickPatch Projects . . . 1496
Differential Releases . . . 1497
Full Releases . . . 1498
Determining the Best Upgrade Solution . . . 1498
Major Upgrade vs. Minor Upgrade vs. Small Update . . . 1500
Automatic Upgrades . . . 1503
Patch vs. QuickPatch Project . . . 1503
Packaging Options for Upgrades . . . 1505
Differential vs. Full Releases . . . 1507
Working with Upgrades, Patches, and QuickPatch Projects . . . 1507
Understanding File Overwrite Rules . . . 1508
Upgrade Considerations . . . 1508
Configuring InstallShield to Automatically Determine the Upgrade Type . . . 1509
Adding an Automatic Upgrade Item . . . 1510
Configuring Automatic Upgrade Item Settings . . . 1510
Converting Automatic Upgrades to Major or Minor Upgrades . . . 1510
Removing Automatic Upgrade Items . . . 1511
Creating Major Upgrades . . . 1511
Adding a Major Upgrade Item . . . 1512
Configuring Major Upgrade Settings . . . 1512
Removing Major Upgrade Items . . . 1513
Creating Minor Upgrades . . . 1513

Adding a Minor Upgrade Item . . . 1513
Configuring Minor Upgrade Settings . . . 1514
Configuring Minor Upgrades to Remove Installed Data . . . 1514
Removing Minor Upgrade Items . . . 1515
Run-Time Behavior for Minor Upgrades . . . 1515
Creating Small Updates . . . 1517
Patching Considerations . . . 1517
Customizing File Properties for the Update Launcher . . . 1519
Specifying the Icon for the Update Launcher . . . 1520
Patch Sequencing . . . 1521
Patch Uninstallation . . . 1524
Non-Administrator Patches . . . 1525
Creating and Applying Patches . . . 1526
Adding a New Patch Configuration Item . . . 1526
Adding a New Latest Setup Item . . . 1526
Adding a New Previous Setup Item . . . 1527
Adding an External File . . . 1527
Configuring Patch Settings . . . 1527
Specifying Identification Information for a Patch . . . 1528
Enabling the Uninstallation of a Patch . . . 1528
Sequencing Patches . . . 1529
Signing a Patch Package . . . 1530
Password-Protecting a Patch Package . . . 1530
Specifying Whether to Build an Update.exe Update Launcher for a Patch . . . 1531
Copying a Latest Setup Item to Another Patch Configuration . . . 1531
Removing Patch Configuration Items . . . 1532
Removing Latest Setup Items . . . 1532
Removing a Previous Setup Item . . . 1532
Building a Patch . . . 1533
Applying Patches . . . 1533
Creating a QuickPatch Project . . . 1534
Creating a QuickPatch Project for an Existing QuickPatch . . . 1535
Specifying Whether to Streamline the QuickPatch Package . . . 1535
Specifying Target Releases for Patching . . . 1536
Specifying Custom Actions for the QuickPatch to Execute . . . 1537
Adding Files to a QuickPatch . . . 1537
Specifying Identification Information for a QuickPatch . . . 1538
Enabling the Uninstallation of a QuickPatch . . . 1538
Sequencing QuickPatch Packages . . . 1538
Signing a QuickPatch Package . . . 1539
Password-Protecting a QuickPatch Package . . . 1539
Specifying Whether to Build an Update.exe Update Launcher for a QuickPatch Package . . . 1540
Deleting Files from the Files To Patch Explorer . . . 1541
Modifying and Removing Installed Files with a QuickPatch . . . 1541
Adding, Modifying, and Deleting Registry Data with a QuickPatch . . . 1541
Validating Upgrades, Patches, and QuickPatch Packages . . . 1542

Validators . . . 1542
Val0001 . . . *1543*
Val0002 . . . *1544*
Val0003 . . . *1545*
Val0004 . . . *1547*
Val0005 . . . *1548*
Val0006 . . . *1549*
Val0007 . . . *1549*
Val0008 . . . *1550*
Val0009 . . . *1552*
Val0011 . . . *1553*
Val0012 . . . *1553*
Val0013 . . . *1554*
Val0014 . . . *1555*
Val0015 . . . *1556*
Patching Assemblies in the Global Assembly Cache . . . 1558
Working with Differential and Full Releases . . . 1559
Creating an InstallScript Release to Update Previous Versions . . . 1559
Notifying End Users about Upgrades Using FlexNet Connect . . . 1561

9 Additional Installation Options . . . 1563

Creating Multilingual Installations . . . 1565
Globalization Tips . . . 1565
Setting the Default Project Language . . . 1566
Settings for Languages . . . 1566
Selecting the Installation Languages . . . 1569
Marking Components as Language Dependent . . . 1570
Installing Components Based on Language . . . 1570
Including Languages in the Release . . . 1571
Translating String Entries . . . 1572
Exporting and Importing String Entries . . . 1572
Modifying Dialogs for Each Language . . . 1574
How an Installation Determines Which Language to Use for the User Interface . . . 1575
Customizing Language Support . . . 1576
Language Identifiers . . . 1576

Installing Multiple Instances of Products . . . 1579
Run-Time Requirements for Multiple-Instance Support . . . 1579
Configuring Multiple Instances in InstallShield . . . 1580
Adding an Instance to a Product Configuration . . . 1581
Naming an Instance . . . 1581
Updating the Product Code of One or More Instances . . . 1582
Setting Properties for an Instance . . . 1583
Deleting an Instance from a Product Configuration . . . 1584
Special Considerations for Multiple-Instance Support . . . 1584
Configuring and Building a Release that Includes Multiple-Instance Support . . . 1585
Configuring and Building a Major Upgrade that Includes Multiple-Instance Support . . . 1586

Creating Patches for Multiple Instances of a Product . . . 1587
Run-Time Behavior for Installing Multiple Instances of a Product . . . 1587
Detecting Conditions on the Target System. . . . 1591
Building Conditional Statements . . . 1591
Conditional Statement Syntax . . . 1593
Detecting Administrator and Elevated Privileges . . . 1595
Detecting First-Time Installations, Maintenance Mode, and Uninstallation . . . 1596
Triggering Behavior Only During a First-Time Installation . . . 1596
Detecting If the End User Has Selected a Specific Feature . . . 1597
Detecting If the End User Is Running a Particular Operating System . . . 1597
Detecting Whether the Installation Is Being Run on a Virtual Machine . . . 1598
Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects . . . 1601
Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects . . . 1604
Platform Condition Settings . . . 1608
File Exists Condition Settings . . . 1611
File Comparison Condition Settings . . . 1612
Registry Exists Condition Settings . . . 1615
Registry Comparison Condition Settings . . . 1617
Property Comparison Condition Settings . . . 1620
MSI Package Condition Settings . . . 1623
MSI Upgrade Condition Settings . . . 1625
Eligible Package Condition Setting . . . 1627
InstallScript Package Condition Settings . . . 1628
Locale Condition Settings . . . 1630
Suite Installed Condition Settings . . . 1631
UWP App Package Condition Settings . . . 1634
UWP Type Present Condition Settings . . . 1635
Package Operation Condition Settings . . . 1636
Feature Operation Condition Settings . . . 1637
Extension Condition Settings . . . 1638
Mode Condition Settings . . . 1639
Guidelines for Defining Conditions in an Advanced UI or Suite/Advanced UI Project . . . 1639
Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed . 1641
Using Locale Conditions in Advanced UI and Suite/Advanced UI Projects . . . 1643
Creating a Custom Condition for an Advanced UI or Suite/Advanced UI Installation . . . 1645
Creating an Extension Condition DLL for an Advanced UI or Suite/Advanced UI Project . . . 1646
Adding an Extension Condition DLL to an Advanced UI or Suite/Advanced UI Project . . . 1649
Searching for Installed Data . . . 1651
Adding a Predefined System Search . . . 1651
Adding a System Search . . . 1652
Modifying a System Search . . . 1652
Special Considerations for Searching the Registry at Run Time . . . 1653
Publishing a System Search to a Repository . . . 1654
Removing a System Search . . . 1655
Searching for XML Data . . . 1655
Editing Installation and Project Tables . . . 1659

Editing Project Tables . . . 1659
Adding New Tables Through the Direct Editor . . . 1660
Adding Records to a Table . . . 1661
Finding and Replacing in the Direct Editor . . . 1661
Editing a Table Record . . . 1662
Editing Fields in a Table . . . 1663
Exporting Tables from the Direct Editor . . . 1664
Importing Tables with the Direct Editor . . . 1665
Deleting Records from a Table . . . 1666
Removing Tables Through the Direct Editor . . . 1666
In-Place Windows Installer Database and Project File Differencing . . . 1667
Entering Binary Data in a Table . . . 1668
Orphaned Directory Entries . . . 1669
Using InstallShield Tools . . . 1671
InstallShield MSI Tools . . . 1671
InstallShield MSI Diff . . . 1671
Determining the Changes that a Transform (.mst File) Makes . . . 1672
Determining Whether a Patch Is Valid for an .msi Package . . . 1672
Determining the Run-Time Effects of a Patch . . . 1673
Determining the Differences Between Two .msi Packages . . . 1673
Running InstallShield MSI Diff from the Command Line to Generate a Log File of Differences . . . 1674
Determining the Differences Between Two InstallShield Projects . . . 1674
InstallShield MSI Query . . . 1674
Running a SQL Query for a Specific Windows Installer Database . . . 1675
InstallShield MSI Sleuth . . . 1675
Viewing Details about an .msi Package that Was Run on a Target System . . . 1675
Searching for All Installed Packages that Include a Specific Component . . . 1676
InstallShield MSI Grep . . . 1676
Searching .msi Packages for Specific Text . . . 1676
InstallShield Cabinet and Log File Viewer . . . 1677
Viewing InstallScript Cabinet Files (.cab) and Header Files (.hdr) . . . 1677
Overview of InstallScript .cab and .hdr Files . . . 1678
Opening InstallScript .cab and .hdr Files . . . 1679
Extracting a File from an InstallScript .cab or .hdr File . . . 1680
Generating a Report about an InstallScript .cab or .hdr File . . . 1680
Viewing InstallScript Log Files (.ilg) . . . 1681
Overview of InstallScript Log Files . . . 1681
Opening InstallScript Log Files . . . 1682
Searching InstallScript Log Files . . . 1682
Converting an InstallScript Log File to a Text File . . . 1683
Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties . . . 1685
Overview of Windows Installer Properties . . . 1685
Windows Installer Property Reference . . . 1687
SETUPEXEDIR . . . 1708
Advanced UI and Suite/Advanced UI Property Reference . . . 1709
Creating Properties in Windows Installer–Based Projects . . . 1714

Creating Properties in Advanced UI and Suite/Advanced UI Projects 1716
Changing an Existing Property 1716
Creating a Localizable Property 1717
Making an Existing Property Localizable 1718
Preventing a Property Value from Being Written in Windows Installer Log Files 1718
Preventing a Property Value from Being Written in Advanced UI and Suite/Advanced UI Debug Log Files 1719
Specifying that a Public Property Should Be a Restricted Public Property 1720
Getting or Setting Windows Installer Properties in InstallScript 1721
Removing a Value from a Property 1721
Deleting a Property 1722
Directly Editing .msi and .msm Databases 1723
Opening Windows Installer Packages 1723
Editing .msi and .msm Databases in Direct Edit Mode 1723
Adding Files in Direct Edit Mode 1724
Adding Merge Modules in Direct Edit Mode 1724

10 Integrating InstallShield with External Applications 1725

Using Source Code Control 1725
Using Source Code Control Integration 1726
Adding Projects to Source Control 1726
Checking Projects out of Source Control 1727
Checking Projects in to Source Control 1727
Unlinking Projects from Source Control 1728
Integrating with Microsoft Visual Studio 1728
Creating InstallShield Projects in Microsoft Visual Studio 1729
Opening InstallShield Projects in Microsoft Visual Studio 1729
Using the VSSolutionFolder Path Variable with Visual Studio Solutions 1729
Adding References to Visual Studio Solutions 1730
Using the Toolbox to Add Dialog Controls 1731
Adding InstallShield Toolbars or Commands to the Visual Studio Toolbar 1731
Building Releases in Microsoft Visual Studio 1732
Adding .NET Assemblies to Installation Projects 1734
Adding Project Output from a Web Service or Web Application Project 1735
Integrating with Microsoft Visual Studio Team Foundation Server 1735
Adding InstallShield Projects to Team Explorer 1736

11 Automating Installation Development and Build Processes 1737

Linking Subfolders to Features Using VBScript 1737
Exporting and Importing String Entries Using the Automation Interface 1740
Running Project Reports Using the Automation Interface 1742
Using Source Code Control at the Command Line 1743
Using Build Status Events 1745
Using the Automation Interface on a 64-Bit System 1746

12 Reference 1747
Menu, Toolbar, and Window Reference 1749
Menus 1749
File Menu 1749
Edit Menu 1750
View Menu 1751
Go Menu 1752
Project Menu 1754
Build Menu 1756
Tools Menu 1760
Window Menu 1762
Help Menu 1762
Toolbars 1763
Standard Toolbar 1763
Controls Toolbar 1765
Layout Toolbar 1767
MSI Debugger Toolbar 1769
Output Window 1769
Dialog Box Reference 1771
.NET 1.1/2.0 Core Language Dialog Box 1775
.NET 1.1/2.0 Language Packs Dialog Box 1775
.NET Installer Class Arguments Dialog Box 1775
Add an Argument/Value Pair Dialog Box 1776
Add Existing Media Dialog Box 1777
Add Files for This Package Dialog Box 1777
Add MIME Type Dialog Box 1778
Add New Method Dialog Box 1779
Add New Property Dialog Box 1779
Add Predefined Search Dialog Box 1780
Add to Source Control Dialog Box 1781
Application Extension Mapping Dialog Box 1781
Application Mappings Dialog Box 1782
Associate Components Dialog Box 1783
Associate DIMs Dialog Box 1783
Batch Build Dialog Box 1783
Browse for Directory Dialog Box 1784
Browse for Directory/Shortcut Target Dialog Box 1785
Browse for File Dialog Box 1787
Browse for File to Run Dialog Box 1788
Browse for Shortcut Icon Dialog Box 1788
Browse for Tile Target Dialog Box 1789
Certificate Selection Dialog Box 1789
Check In Dialog Box 1791
Check Out Dialog Box 1792
Component Properties Dialog Box 1792
General Tab 1793

File Linking Tab . . . 1795
Features Tab . . . 1796
Condition Builder Dialog Box . . . 1796
Content Source Path Dialog Box . . . 1798
Convert InstallScript MSI Dialog Box . . . 1798
Create a New Component Dialog Box . . . 1800
Create a New Feature Dialog Box . . . 1800
Custom Error Handling Dialog Box . . . 1801
Custom Errors Dialog Box . . . 1801
Customize Validation Settings Dialog Box . . . 1802
Data Languages Dialog Box . . . 1802
Delete Property Dialog Box . . . 1802
Dependencies Dialog Box . . . 1803
Dialog Images Dialog Box . . . 1803
Digitally Sign Setup Dialog Box . . . 1804
Differences Dialog Box . . . 1805
Dynamic File Link Settings Dialog Box . . . 1805
Edit Data Dialog Box . . . 1807
Edit IIS Metabase Value Dialog Box . . . 1807
Edit List Options Dialog Box . . . 1808
Edit Registry Data Dialog Box . . . 1809
Edit Virtual Machine Configurations Dialog Box . . . 1810
Error Mapping Properties Dialog Box . . . 1812
Exclusion Languages Dialog Box . . . 1813
Export Tables Dialog Box . . . 1813
Feature Condition Builder Dialog Box . . . 1813
File Details Dialog Box . . . 1815
File Properties Dialog Box . . . 1815
File Properties Dialog Box (InstallScript Installation Projects) . . . 1819
File Removal Properties Dialog Box . . . 1819
Find and Replace Dialog Box . . . 1821
Find String ID in Project Dialog Box . . . 1823
Folder Properties Dialog Box . . . 1823
General Tab . . . 1824
Sharing Tab . . . 1824
Function Signature Dialog Box . . . 1825
General Options - Advanced Dialog Box . . . 1828
General Options - Other Disk Files Dialog Box . . . 1829
History Dialog Box . . . 1830
Import Dialog Dialog Box . . . 1830
Import InstallScript Files Dialog Box . . . 1830
Import SQL Script Files Dialog Box . . . 1831
Insert Action Dialog Box . . . 1831
InstallShield Prerequisite Properties Dialog Box . . . 1834
Languages Dialog Box . . . 1836
Link Type Dialog Box . . . 1836

Logging Options for Windows Installer 4.0 and Later Dialog Box. . . . 1838
Lowercase Component Directory Warning . . . 1839
Media File Properties Dialog Box . . . 1839
Merge Module Configurable Values Dialog Box . . . 1840
Merge Module Properties Dialog Box . . . 1840
Media Tab . . . 1841
Destination Tab . . . 1841
Method Browser Dialog Box . . . 1842
Method Signature Dialog Box . . . 1842
MIME Types Dialog Box . . . 1844
Modify Dynamic Links Dialog Box. . . . 1845
Modify Property Dialog Box/Release Wizard—Platforms Panel . . . 1846
Module Dependencies Dialog Box . . . 1848
Module Exclusions Dialog Box. . . . 1848
MSI Value Dialog Box . . . 1849
MST SIS Settings Dialog Box . . . 1849
Multi-Line String Value Dialog Box. . . . 1850
New Dependency Dialog Box . . . 1853
New File Dialog Box. . . . 1853
New Project Dialog Box . . . 1853
Operating Systems Dialog Box . . . 1854
Options Dialog Box . . . 1855
General Tab. . . . 1856
File Locations Tab . . . 1856
Path Variables Tab. . . . 1857
User Interface Tab . . . 1858
Preferences Tab . . . 1859
Updates Tab . . . 1860
.NET Tab . . . 1861
Files View Tab. . . . 1861
File Extensions Tab . . . 1863
Prerequisites Tab . . . 1865
Source Control Tab . . . 1865
Directory Tab . . . 1866
Resource Tab . . . 1867
Validation Tab . . . 1868
Repository Tab. . . . 1868
SQL Scripts Tab . . . 1869
Merge Modules Tab . . . 1870
Quality Tab . . . 1871
Other IIS Properties Dialog Box. . . . 1871
Other Window Styles Dialog Box . . . 1871
Other Window Styles for Dialogs . . . 1872
Other Window Styles for Bitmap, Icon, and Text Area Controls . . . 1873
Other Window Styles for Button Controls. . . . 1875
Other Window Styles for Combo Box Controls . . . 1877

Other Window Styles for List View Controls . . . 1878
Other Window Styles for Edit Field and List Box Controls . . . 1880
Other Window Styles for Line Controls . . . 1882
Other Window Styles for Progress Bar Controls . . . 1884
Other Window Styles for Selection Tree Controls . . . 1884
Outputs Dialog Box . . . 1886
Overwrite Dialog Box . . . 1886
Patch Sequence Dialog Box . . . 1888
Path Variable Overrides Dialog Box . . . 1889
Path Variable Recommendation Dialog Box . . . 1889
Permissions Dialog Boxes for Files and Directories . . . 1890
Permissions Dialog Boxes for Registry Keys . . . 1892
Platform Suites Dialog Box . . . 1894
Platforms Dialog Box . . . 1895
Postbuild Event Dialog Box . . . 1897
Prebuild Event Dialog Box . . . 1898
Precompression Event Dialog Box . . . 1898
Prerequisite Condition Dialog Box . . . 1898
Privileges Dialog Box . . . 1905
Product Condition Builder Dialog Box . . . 1905
Project Settings Dialog Box . . . 1907
Project Tab . . . 1907
Application Tab (in Windows Installer–Based Projects) . . . 1907
Application Tab (in InstallScript Projects) . . . 1909
Platforms Tab . . . 1911
Languages Tab . . . 1912
Maintenance Tab . . . 1912
Language-Independent Object Properties Tab . . . 1913
Language-Specific Object Properties Tab . . . 1913
Rename Key Dialog Box . . . 1914
Required Features Dialog Box . . . 1915
Resolve Conflict Dialog Box . . . 1915
Save As Dialog Box . . . 1916
Script Conversion Dialog Box . . . 1917
Script Editor Properties Dialog Box . . . 1918
SCM Pre-shutdown Delay Dialog Box . . . 1921
Security Descriptor Dialog Box . . . 1922
Select File Dialog Box . . . 1922
Select Media Location Dialog Box . . . 1923
Select SQL Server Dialog Box . . . 1923
Select String Dialog Box . . . 1923
Serial Number Violation Dialog Box . . . 1925
Server Locations Dialog Box . . . 1925
Service Options Dialog Box for a Time Delay of an Auto-Start Service . . . 1926
Service Options Dialog Box for Running Recovery Actions . . . 1927
Service SID Dialog Box . . . 1927

Set File(s) URL Dialog Box. 1928
Settings Dialog Box 1928
Compile/Link Tab. 1929
Run/Debug Tab 1933
MSI Log File Tab 1934
Setup Languages Dialog Box 1934
SQL Server Requirement Dialog Box 1935
Supersedence Dialog Box. 1936
UI Languages Dialog Box 1937
Update Merge Module Search Path Dialog Box. 1937
Wizard Reference 1939
Component Wizard 1940
Welcome Panel 1941
Setup Best Practices 1942
Destination Panel. 1943
Files Panel 1943
Summary Panel 1944
Component Type Panel 1945
COM Server Component Type 1946
COM Server File Panel 1947
Classes Panel 1949
Context Types Panel 1950
Type Library Panel 1950
AppID General Information Panel. 1951
AppID Advanced Information Panel. 1952
Summary Panel 1953
Control Service Component Type. 1953
Specify Service Panel. 1954
Installation Events Panel. 1955
Uninstallation Events Panel 1956
Wait Type Panel 1957
Summary Panel 1957
Install Service Component Type. 1958
Service Executable Panel 1959
Service Type Information Panel. 1960
Service Start Type Information Panel 1961
Service Load Order Panel. 1961
Error Control Panel 1962
Service Logon Panel 1962
Summary Panel 1963
Font Component Type. 1964
Add Installed Fonts Panel. 1964
Add New Fonts Panel 1965
Summary Panel 1966
Component Wizard General Summary Panel 1966
Convert Source Paths Wizard. 1967
Welcome Panel 1967
Search Panel 1967

Search Results and Recommendations Panel . . . 1967
Updating Your Project Panel . . . 1968
Update Completed Panel . . . 1968
Create New QuickPatch Wizard . . . 1968
Welcome Panel . . . 1969
QuickPatch Project Base Panel . . . 1969
Original Setup Package Panel . . . 1970
Location of Existing QuickPatch Panel . . . 1971
Create Table Wizard . . . 1972
Welcome Panel . . . 1972
Column Properties Panel . . . 1973
Summary Panel . . . 1974
Custom Action Wizard . . . 1974
Welcome Panel . . . 1975
Basic Information Panel . . . 1976
Action Type Panel . . . 1976
Function Definition Panel . . . 1981
Action Parameters Panel . . . 1984
Managed Method Definition Panel . . . 1990
In-Sequence Scripts Panel . . . 1992
Additional Options Panel . . . 1992
Respond Options Panel . . . 1994
Insert into Sequence Panel . . . 1995
Summary Panel . . . 1996
Database Import Wizard . . . 1996
Welcome Panel . . . 1996
SQL Server Panel . . . 1997
SQL Database Panel . . . 1997
Database Tables Panel . . . 1998
Objects to Script Panel . . . 1998
Scripting Options Panel . . . 1998
Advanced Scripting Options Panel . . . 2001
Summary Panel . . . 2002
Device Driver Wizard . . . 2003
Welcome Panel . . . 2003
Device Driver Package Panel . . . 2004
Device Driver Files Panel . . . 2004
Device Driver Information Panel . . . 2004
Device Driver Type . . . *2005*
Project-Wide Device Driver Information Panel . . . 2005
Summary Panel . . . 2007
Dialog Wizard . . . 2007
Welcome Panel . . . 2007
Dialog Template Panel . . . 2007
User Interface Sequence Panel . . . 2010
Dialog Position and Condition Panel . . . 2010

DirectX Object Wizard . . . 2011
Welcome Panel . . . 2011
Object Settings Panel . . . 2011
Summary Panel . . . 2012
Dynamic Scanning Wizard . . . 2012
Welcome Panel . . . 2013
Filter Files Panel . . . 2013
Specify the Executable Panel . . . 2014
Specify Application File Panel . . . 2014
Launch the Application . . . 2015
Your Application is Running Panel . . . 2015
File Selection Panel . . . 2016
Scan Results Panel . . . 2016
Completing the Dynamic Scanning Wizard Panel . . . 2016
Export Components Wizard . . . 2017
Welcome Panel . . . 2018
Select Components Panel . . . 2018
Select Target File Panel . . . 2019
Target File Info Panel . . . 2019
Summary Panel . . . 2020
Function Wizard . . . 2020
Function Name Panel . . . 2020
Function Parameters Panel . . . 2021
Import DIM Wizard . . . 2021
Welcome Panel . . . 2022
Select Source File Panel . . . 2022
Import Panel . . . 2022
Import REG File Wizard . . . 2023
Welcome Panel . . . 2023
Import Registry File Panel . . . 2023
Import Conflict Options Panel . . . 2024
Import Progress Panel . . . 2024
Import XML Settings Wizard . . . 2024
Welcome Panel . . . 2025
XML File Panel . . . 2025
XML Element Panel . . . 2025
XML Progress Panel . . . 2026
XML Results Panel . . . 2026
Microsoft .NET Framework Object Wizard . . . 2026
Step 1 Panel . . . 2027
Step 2 Panel . . . 2027
New Language Wizard . . . 2029
Welcome Panel . . . 2029
Project Language Panel . . . 2030
Project Files Panel . . . 2030
Summary Panel . . . 2031

Open MSI/MSM Wizard 2032
Welcome Panel 2032
File Locations Panel 2033
Open MSP Wizard 2033
Welcome Panel 2033
Base MSI Package Panel 2033
Open Transform Wizard 2034
Welcome Panel 2035
Transform Information Panel 2035
Additional Transforms Panel 2035
Create a Response Transform Panel 2036
Publish Wizard 2037
Welcome Panel 2038
Repository Panel 2038
Publishing Information Panel 2038
Summary Panel 2038
Redistributable Downloader Wizard 2038
Reg-Free COM Wizard 2039
Welcome Panel 2039
Select an EXE File Panel 2040
Select the Manifest File Panel 2040
Select Files Panel 2040
Summary Panel 2040
Release Wizard 2041
Welcome Panel 2041
Product Configuration Panel 2041
Specify a Release Panel 2042
Filtering Settings Panel 2042
Merge Module Options Panel 2043
Setup Languages Panel 2043
Media Type Panel 2045
Disk Spanning Options Panel 2046
Disk Spanning Advanced Settings Panel 2046
Web Type Panel 2048
Downloader Options Panel 2049
Install From The Web Options Panel 2050
One-Click Install Panel 2050
Local Machine Panel 2051
Release Configuration Panel 2052
Custom Compression Settings Panel 2053
Setup Launcher Panel 2053
Windows Installer Location Panel 2054
InstallShield Prerequisites Panel 2055
Digital Signature Panel 2057
Digital Signature Options Panel 2057
Password & Copyright Panel 2061

.NET Framework Panel . . . 2062
.NET Run-Time Options Panel . . . 2064
.NET Language Pack Run-Time Options Panel . . . 2064
Visual J# Run-Time Options Panel . . . 2064
Advanced Settings Panel . . . 2065
Release Settings Summary Panel . . . 2068
General Options Panel . . . 2068
Features Panel . . . 2070
Media Layout Panel . . . 2070
Custom Media Layout Panel . . . 2071
User Interface Panel . . . 2072
Internet Options Panel . . . 2073
Update Panel . . . 2074
Object Difference Dialog Box . . . 2076
Postbuild Options Panel . . . 2076
Setup Best Practices Wizard . . . 2078
Welcome Panel . . . 2078
Compliance Panel . . . 2078
Static Scanning Wizard . . . 2079
Welcome Panel . . . 2080
Filter Files Panel . . . 2080
Scanning Progress Panel . . . 2080
File Selection Panel . . . 2081
Scan Results Panel . . . 2081
Completing the Static Scanning Wizard Panel . . . 2082
System Search Wizard . . . 2082
Welcome Panel . . . 2083
What do you want to find? Panel . . . 2083
How do you want to look for it? Panel (Defining Your System Search Method) . . . 2084
Specify the file details Panel (File Search Options) . . . 2086
How do you want to look for it? Panel (Folder Search Options) . . . 2087
How do you want to look for it? Panel (Specific Folder Options) . . . 2087
How do you want to look for it? Panel (Registry Search Options) . . . 2088
How do you want to look for it? Panel (.ini File Search Options) . . . 2089
How do you want to look for it? Panel (Component Search Option) . . . 2090
Specify the data that you wish to find in the XML file Panel (XML File Options) . . . 2090
What do you want to do with the value? Panel . . . 2091
What do you want to do with the value? Panel . . . 2092
Transform Wizard . . . 2093
Welcome Panel . . . 2094
Specify Files Panel . . . 2094
Validation Settings Panel . . . 2094
Suppress Error Conditions Panel . . . 2096
Specify Output File Name Panel . . . 2097
Summary Panel . . . 2097
Completing the Transform Wizard Panel . . . 2097

Upgrade Validation Wizard . . . 2098
Welcome Panel . . . 2098
Settings Panel . . . 2099
Summary Panel . . . 2099
User Interface Wizard . . . 2100
Predefined Task Pages Panel . . . 2100
Task Configuration Panel . . . 2103
Visual Studio .NET Wizard for Visual Basic .NET, C# .NET, and Visual C++ .NET . . . 2104
Welcome Panel . . . 2104
Solution Panel . . . 2105
Visual Studio Deployment Project Import Wizard . . . 2105
Welcome Panel . . . 2105
Project File Panel . . . 2106
Options Panel . . . 2106
Summary Panel . . . 2110
View Reference . . . 2111
Installation Information View . . . 2111
General Information View . . . 2112
General Information View Settings . . . 2113
Update Notifications View . . . 2150
Organization View . . . 2150
Setup Design View . . . 2153
Features View . . . 2154
Feature Settings . . . 2155
Components View . . . 2167
Component Settings . . . 2168
Advanced Settings for a Component . . . 2188
Application Path Settings . . . 2189
Assembly Settings . . . 2190
COM Registration Settings . . . 2192
File Type Settings . . . 2197
Services Settings . . . 2199
Publishing Settings . . . 2200
Device Driver Settings . . . 2201
Other Data Settings . . . 2204
Setup Types View . . . 2204
Packages View . . . 2206
Common Tab . . . 2207
Features Tab . . . 2236
Dynamic Link Settings . . . 2236
DIM References View . . . 2238
General Tab . . . 2239
Instructions Tab . . . 2239
Build Options Tab . . . 2240
Features Tab . . . 2240
Sequences Tab . . . 2241
Application Data View . . . 2241

Files and Folders View. 2242
Destination Folders *2244*
Redistributables View 2247
Prerequisites View 2250
Objects View. 2252
System Configuration View 2253
Shortcuts View 2257
Shortcut Settings *2259*
Folder Settings *2275*
Tile Configuration Settings *2278*
Registry View 2281
ODBC Resources View 2282
ODBC Resource Settings *2283*
INI File Changes View 2285
.ini File Settings *2286*
Section Settings for an .ini File *2287*
Keyword Settings for an .ini File. *2288*
Environment Variables View 2289
Environment Variable Settings. *2289*
XML File Changes View. 2291
General Tab for an XML File. *2293*
Advanced Tab for an XML File *2293*
Namespace Tab for an XML File *2294*
General Tab for an XML Element. *2295*
Advanced Tab for an XML Element *2296*
Text File Changes View. 2297
Change Set Settings. *2298*
Change Settings *2300*
Scheduled Tasks View. 2302
Scheduled Tasks Settings. *2303*
Services View 2306
Services View Settings *2307*
Server Configuration View 2320
Internet Information Services View. 2321
Web Site Settings *2322*
Application and Virtual Directory Settings *2330*
Application Pool Settings *2337*
Web Service Extension Settings *2343*
Component Services View 2344
Installation Tab. *2345*
SQL Scripts View 2347
General Tab *2348*
Requirements Tab *2351*
Advanced Tab. *2353*
SQL Script Level *2356*
General Tab. 2357

Script Tab . . . 2358
Runtime Tab . . . 2358
Database Import Tab . . . 2361
Text Replacement Tab . . . 2361
Behavior and Logic View . . . 2361
InstallScript View . . . 2364
Script Editor in the InstallScript View . . . *2366*
Custom Actions and Sequences View (or Custom Actions View) . . . 2367
Custom Action Types . . . *2369*
Custom Action Settings . . . *2373*
Action Text Settings . . . *2384*
Support Files View (Advanced UI, Basic MSI, InstallScript Object, and Suite/Advanced UI Projects) . . . 2386
Support Files/Billboards View (InstallScript and InstallScript MSI Projects) . . . 2388
System Search View . . . 2389
Property Manager View . . . 2390
Events View . . . 2393
Events View Settings . . . *2394*
User Interface View . . . 2401
Dialogs View (InstallScript, InstallScript MSI, and InstallScript Object Projects) . . . 2403
Dialogs View (Basic MSI and Other Windows Installer–Based Projects) . . . 2403
Wizard Interface View . . . 2404
Wizard Interface View Toolbar . . . *2406*
Billboards View . . . 2410
Billboard Settings . . . *2411*
Settings for Adobe Flash Application File Billboards and Image Billboards . . . *2411*
String Editor View . . . 2415
Media View . . . 2417
Path Variables View . . . 2419
Upgrades View . . . 2424
Common Tab . . . *2426*
Advanced Tab . . . *2427*
Major Upgrade Item: Common Tab . . . *2429*
Major Upgrade Item: Advanced Tab . . . *2431*
Minor Upgrade/Small Update Item Settings . . . *2435*
Automatic Upgrade Item Settings . . . *2435*
Releases View . . . 2436
Product Configuration Settings . . . *2436*
General Tab for a Product Configuration . . . 2437
Multiple Instances Tab for a Product Configuration . . . 2446
Release Settings . . . *2447*
Build Tab for a Release . . . 2447
Setup.exe Tab for a Release . . . 2459
Signing Tab for a Release . . . 2473
.NET/J# Tab for a Release . . . 2478
Internet Tab for a Release . . . 2482
Events Tab for a Release . . . 2485
Windows App Tab for a Release . . . 2494

Chained .msi Package Settings *2501*
Patch Design View 2505
Patch Configuration *2505*
Common Tab 2506
Identification Tab 2507
Digital Signature Tab 2508
Sequence Tab 2509
Advanced Tab 2510
Latest Setup *2518*
Previous Setups *2518*
Common Tab 2519
Advanced Tab 2519
Additional External Files *2521*
External File 2521
Additional Tools View 2521
Dependency Scanners 2523
MSI Debugger 2523
Direct Editor 2524
Windows Installer Table Reference *2530*
InstallShield Table Reference *2530*
ISAlias Table 2531
ISCOMCatalogAttribute Table 2532
ISCOMCatalogCollection Table 2533
ISCOMCatalogCollectionObject Table 2533
ISCOMCatalogObject Table 2534
ISCOMPlusApplication Table 2534
ISCustomActionReference Table 2535
ISIISItem Table 2536
ISIISProperty Table 2536
ISProductConfigurationInstance Table 2538
ISSelfReg Table 2539
InstallShield Columns in Standard Windows Installer Tables *2540*
QuickPatch Projects 2542
Patch Settings View 2542
General Information View *2543*
Product Properties 2543
Build Settings 2543
Common Tab 2544
Identification Tab 2544
Digital Signature Tab 2545
Advanced Tab 2546
History 2552
Custom Action 2552
Files View *2552*
New File Settings 2552
Modified/Deleted File Settings 2553
Registry View *2555*
InstallShield Prerequisite Editor Reference **2557**

Properties Tab. . . . 2558
Conditions Tab . . . 2558
Files to Include Tab . . . 2559
Application to Run Tab . . . 2559
Behavior Tab . . . 2561
Dependencies Tab . . . 2564
Errors and Warnings . . . 2567
Build Errors and Warnings . . . 2568
Media Build Errors and Warnings . . . 2654
Error 118 . . . 2655
Error 129 . . . 2655
Warning -5000 . . . 2656
Warning -5006 . . . 2657
Warning -5020 . . . 2657
Warning -5021 . . . 2657
Warning -5032 . . . 2657
Warning -5050 . . . 2658
Warning -5051 . . . 2658
Error -7040 . . . 2658
Error -7041 . . . 2658
Error -7043 . . . 2659
Error -7044 . . . 2659
Error -7045 . . . 2659
Error -7126 . . . 2659
Error -7127 . . . 2660
Error -7275 . . . 2660
Error -7276 . . . 2660
Error -7380 . . . 2661
Error -9008 . . . 2661
MSI/MSM Conversion Errors . . . 2661
Windows Installer Run-time Errors . . . 2665
Setup.exe Return Values and Run-Time Errors (InstallScript Projects) . . . 2674
Setup.exe Return Values and Run-Time Errors (Basic MSI and InstallScript MSI Projects) . . . 2677
Setup.exe Return Values and Run-Time Errors (Advanced UI and Suite/Advanced UI Projects) . . . 2681
Visual Studio Project Import Errors and Warnings. . . . 2689
Upgrade Errors (Upgrading from InstallShield Professional) . . . 2704
Upgrade Warnings (Upgrading from InstallShield Professional). . . . 2712
Upgrade Warning 289: File Link Does Not Exist for Dynamically Linked File . . . 2712
Upgrade Warning -305: Include In Build Property for the InstallShield Professional Component Could Not Be Directly Migrated . . . 2713
Upgrade Warning -480: Incrementing the Shared .dll Reference Count for Files in a File Group. . . . 2713
Upgrade Warning -481: Incrementing the Shared DLL Reference Count for Dynamically Linked Files in a File Group . . . 2714
Upgrade Warning -486: Specifying Shortcut Icons . . . 2714
Upgrade Warning -487: Overwriting Files Upon Installation. . . . 2715
Upgrade Warning -488: Obsolete Event Encountered in Script File. . . . 2716
Upgrade Warning 495: ODBC Core Not Installed by Default . . . 2717

Upgrade Errors and Warnings (Upgrading from InstallShield—Windows Installer Edition) . . . 2717
HRESULT Values for Windows Installer Run-time Errors . . . 2718
DIFxAPI Errors (InstallScript Projects) . . . 2719
"String PRODUCT_NAME was not found in string table" Error . . . 2721
InstallScript Error Information . . . 2722
InstallScript Syntax or Compiler Errors . . . 2722
Error C8001 . . . *2726*
Error C8002 . . . *2727*
Error C8003 . . . *2727*
Error C8004 . . . *2728*
Error C8005 . . . *2728*
Error C8006 . . . *2728*
Error C8007 . . . *2729*
Error C8008 . . . *2729*
Error C8009 . . . *2729*
Error C8010 . . . *2730*
Error C8011 . . . *2730*
Error C8012 . . . *2730*
Error C8013 . . . *2731*
Error C8014 . . . *2731*
Error C8015 . . . *2731*
Error C8016 . . . *2732*
Error C8017 . . . *2732*
Error C8018 . . . *2733*
Error C8019 . . . *2733*
Error C8020 . . . *2733*
Error C8021 . . . *2734*
Error C8022 . . . *2734*
Error C8023 . . . *2734*
Error C8024 . . . *2735*
Error C8025 . . . *2735*
Error C8026 . . . *2736*
Error C8027 . . . *2736*
Error C8028 . . . *2736*
Error C8031 . . . *2737*
Error C8032 . . . *2737*
Error C8033 . . . *2737*
Error C8034 . . . *2738*
Error C8035 . . . *2738*
Error C8036 . . . *2738*
Error C8037 . . . *2739*
Error C8038 . . . *2739*
Error C8039 . . . *2739*
Error C8040 . . . *2740*
Error C8042 . . . *2740*
Error C8043 . . . *2740*

Error C8044. . . . . *2741*
Error C8045. . . . . *2741*
Error C8046. . . . . *2741*
Error C8047. . . . . *2741*
Error C8048. . . . . *2742*
Error C8049. . . . . *2742*
Error C8050. . . . . *2742*
Error C8051. . . . . *2743*
Error C8052. . . . . *2743*
Error C8053. . . . . *2743*
Error C8054. . . . . *2743*
Error C8055. . . . . *2744*
Error C8057. . . . . *2744*
Error C8058. . . . . *2744*
Error C8059. . . . . *2745*
Error C8062. . . . . *2745*
Error C8063. . . . . *2745*
Error C8064. . . . . *2746*
Error C8065. . . . . *2746*
Error C8066. . . . . *2746*
Error C8067. . . . . *2746*
Error C8068. . . . . *2747*
Error C8069. . . . . *2747*
Error C8070. . . . . *2747*
Error C8071. . . . . *2748*
Error C8072. . . . . *2748*
Error C8073. . . . . *2748*
Error C8074. . . . . *2749*
Error C8075. . . . . *2749*
Error C8076. . . . . *2749*
Error C8077. . . . . *2750*
Error C8078. . . . . *2750*
Error C8079. . . . . *2750*
Error C8080. . . . . *2751*
Error C8081. . . . . *2751*
Error C8082. . . . . *2751*
Error C8083. . . . . *2752*
Error C8084. . . . . *2752*
Error C8085. . . . . *2752*
Error C8086. . . . . *2753*
Error C8087. . . . . *2753*
Error C8088. . . . . *2753*
Error C8089. . . . . *2753*
Error C8090. . . . . *2754*
Error C8091. . . . . *2754*
Error C8092. . . . . *2754*

Error C8093 2755
Error C8097 2755
Error C8098 2755
Error C8099 2756
Error C8100 2756
Error C8101 2756
Error C8112 2757
Error C8113 2757
Error C8114 2757
Error C8115 2757
Error C8126 2758
Error C8127 2758
Error C8128 2758
Error C8522 2759
InstallScript Fatal Errors 2759
Error F8501 2760
Error F8502 2761
Error F8503 2761
Error F8504 2761
Error F8505 2762
Error F8506 2762
Error F8508 2763
Error F8509 2763
Error F8510 2763
Error F8511 2764
Error F8512 2764
Error F8513 2764
Error F8514 2765
Error F8515 2765
Error F8516 2765
Error F8517 2765
Error F8518 2766
Error F8519 2766
InstallScript Internal Errors 2766
Error C9001 2766
InstallScript Warnings 2767
Warning C7501 2767
Warning C7502 2767
Warning C7503 2768
Warning C7505 2768
Virtualization Conversion Errors and Warnings 2769
Error -9000: Unknown Exception 2769
Error -9001: Unknown COM 2769
Error -9002: Error Opening Package 2769
Error -9003: Error Saving Package 2770
Error -9004: Process Cancelled By User 2770

Error -9005: Error Creating Temporary Folder . . . 2771
Error -9006: Error Decompressing Package . . . 2771
Error -9007: File With Extension Not Found . . . 2772
Error -9008: Error Extracting Icon . . . 2772
Error -9009: Unknown Provider. . . . 2772
Error -9010: Invalid Target File Name . . . 2773
Error -9011: Error Reading MSI Table . . . 2773
Error -9012: Unexpected Error in Method. . . . 2773
Error -9013: Type Library Not Found . . . 2774
Error -9014: ShellExecute Failed . . . 2774
Error -9015: Unable to Determine Full Path for Driver . . . 2774
Warning -9016: Contents of Table Ignored . . . 2775
Warning -9017: .NET 1.x Assembly Not Supported. . . . 2776
Warning -9018: Custom Actions Ignored . . . 2776
Warning -9019: Conditionalized Components. . . . 2777
Error -9020: Directory With Null Parent. . . . 2778
Error -9021: Unable to Extract COM Data . . . 2778
Error -9022: Complus Table . . . 2779
Error -9024: FileSFPCatalog . . . 2779
Warning -9026: LaunchCondition Table . . . 2779
Warning -9027: LockPermissions Table. . . . 2780
Error -9028: MoveFile Table. . . . 2781
Error -9029: MsiDriverPackages Table . . . 2781
Warning -9030: ODBCTranslator Table . . . 2782
Warning -9031: RemoveFile Table . . . 2782
Warning -9032: RemoveIniFile Table . . . 2783
Warning -9033: RemoveRegistry Table . . . 2783
Error -9036: ISCEInstall Table. . . . 2784
Error -9037: ISComPlusApplication Table . . . 2784
Error -9038: ISPalmApp Table . . . 2785
Error -9039: ISSQLScriptFile Table . . . 2785
Error -9040: ISVRoot Table. . . . 2786
Error -9041: ISXmlFile Table. . . . 2786
Error -9051: Package Decompression Canceled . . . 2787
Error -9100: CreateInstance of Package Object Failed . . . 2787
Error -9101: Create Operation of Package Object Failed . . . 2787
Error -9102: Failed to Write Header Information . . . 2788
Error -9103: Citrix Finalization Failed . . . 2788
Error -9104: Citrix Save Failed . . . 2789
Error -9105: Error Initializing Citrix Writer . . . 2789
Error -9106: Error Initializing Citrix Package . . . 2789
Error -9107: Error Writing Citrix File Entries . . . 2790
Error -9108: Error Determining Source File Path . . . 2790
Error -9109: Error Writing Citrix Folder Entries. . . . 2790
Error -9110: Error Writing Citrix Registry Entries . . . 2791
Error -9113: Error Writing Citrix INI File Entries . . . 2791

Error -9114: Error Writing Citrix Shortcuts . . . 2791
Error -9115: Error Saving Citrix Profile . . . 2792
Error -9116: Error Creating Empty Citrix Profile . . . 2792
Error -9117: Error Creating Intermediate Folder . . . 2792
Error -9118: Error Initializing Citrix Profile . . . 2793
Error -9119: Error Creating Default Target in Citrix Profile . . . 2793
Error -9120: Error Deleting File From Profile . . . 2793
Error -9121: Failed to Copy File into Citrix Profile . . . 2794
Error -9122: Target Does Not Exist in Citrix Profile . . . 2794
Error -9124: No Shortcuts Created for this Profile . . . 2794
Error -9125: Error Writing Citrix File Type Associations . . . 2795
Error -9126: Failed to Sign Profile Using Certificate . . . 2795
Error -9127: Could Not Create Script Execution . . . 2795
Warning -9128: Duplicate Shortcut . . . 2796
Warning -9129: Duplicate Shortcut Names . . . 2796
Warning -9130: Duplicate Shortcut Targets . . . 2797
Warning -9131: Unable to Resolve Installer Variable . . . 2797
Warning -9132: 16 Color Shortcut Icon Not Found . . . 2797
Warning -9133: Shortcut Icon Not Found . . . 2798
Warning -9134: Failure to Extract Icon from Executable . . . 2798
Error -9135: Shortcut Target is 16-Bit . . . 2799
Warning -9136: Some Files May Not Be Decompressed . . . 2799
Warning -9137: Destination Directory Cannot Be Found . . . 2799
Warning -9138: Ignoring a DuplicateFile Table Entry . . . 2800
Warning -9150: Warning Building Windows Installer Package . . . 2801
Error -9151: Error Building Windows Installer Package . . . 2801
Error -9200: ThinApp Must Be Installed . . . 2801
Warning -9201: Extension for Shortcut Files Must Be ".exe" . . . 2802
Error -9202: No Applications Were Created . . . 2802
Error -9203: ThinApp Tool is Missing . . . 2803
Error -9204: Duplicate Shortcut . . . 2803
Error -9205: Identically Named Shortcut Already Exists, But With Different Parameters . . . 2803
Error -9206: Identically Named Shortcut Already Exists, But With a Different Target . . . 2804
Error -9207: Error During Build Process (vregtool.exe) . . . 2804
Error -9208: Error Occurred During Build Process (vftool.exe) . . . 2804
Error -9209: Error Occurred During Build Process (tlink.exe) . . . 2805
Error -9300: Unhandled Exception During AdviseFile Operation . . . 2805
Error -9301: Unhandled Exception During AdviseRegistry Operation . . . 2806
Error -9302: Unhandled Exception During Command Action . . . 2806
Error -9303: Unhandled Exception During Alter File Action . . . 2806
Error -9304: Unhandled Exception During Alter Registry Action . . . 2807
Error -9305: Unhandled Exception During Create Action . . . 2807
Error -9306: Unhandled Exception During Execution of Rules Engine . . . 2807
Error -9401: Error Initializing App-V Writer . . . 2808
Error -9402: Error Initializing App-V Package . . . 2808
Error -9403: Error Writing App-V File Entries . . . 2808

Error -9404: Error Writing App-V Folder Entries. . . . 2809
Error -9405: Error Writing App-V Registry Entries . . . 2809
Error -9406: Error Writing App-V INI File Entries . . . 2809
Error -9407: Error Writing App-V Shortcuts . . . 2810
Error -9408: Error Writing App-V File Type Data . . . 2810
Error -9409: Error Saving App-V Data . . . 2810
Error -9410: Error Determining Source File Path . . . 2811
Error -9411: OSD File Template Could Not Be Extracted . . . 2811
Error -9412: OSD File Could Not Be Saved . . . 2811
Error -9413: App-V OSD Save . . . 2812
Warning -9414: Local App-V Application Specified as a Dependency of the Primary Application. . . . 2812
Error -9415: Dependency Application Was Not Found. . . . 2812
Warning -9416: Invalid Primary Application Directory . . . 2813
Error -9417: Dependency Application's OSD File Contains an Invalid HREF Value . . . 2813
Error -9418: Error While Privatizing Side-By-Side Assemblies. . . . 2814
Error -9419: Error Inserting Watermark . . . 2814
Error -9424: Building an App-V 5.x Package. . . . 2814
Warning -9500: Shortcut Missing . . . 2815
Error -10000: Process Cancelled By User. . . . 2815
Warning -10001: Suite File Missing . . . 2815
Warning -10002: Suite File is Duplicate. . . . 2816
Warning -10003: Application File Missing. . . . 2816
Warning -10004: INI File Missing . . . 2816
Fix 11000: Excluding TCPIP Registry Entries . . . 2817
Fatal Error 11001: Fail on VMware . . . 2817
Warning 11003: Control Panel Applet - Citrix . . . 2817
Fix 11004: Control Panel Applet - ThinApp. . . . 2818
Fatal Error 11005: QuickTime 7.4.1 Causes Fatal Error . . . 2818
Fix 11006: Adobe Distiller Exclude AdobePDFSettings . . . 2818
Fix 11007: Exclude URL Shortcut. . . . 2819
Steps to Take Before Calling Technical Support . . . 2819
Application Features Requiring Pre- or Post-Conversion Actions . . . 2819
Automation Interface . . . 2821
Automation Objects. . . . 2821
ISWiProject Object. . . . 2822
AddAdvancedFile Method . . . 2837
AddAutomaticUpgradeEntry Method . . . 2838
AddComponent Method . . . 2839
AddCustomAction Method . . . 2840
AddFeature Method . . . 2840
AddLanguage Method . . . 2842
AddPathVariable Method . . . 2842
AddProductConfig Method . . . 2843
AddProperty Method . . . 2844
AddSetupFile Method. . . . 2845
AddSetupType Method . . . 2845

AddSQLServerConnection Method 2846
AddUpgradeTableEntry Method 2847
BuildPatchConfiguration Method 2848
BuildPCPFile Method 2850
CloseProject Method 2851
CreatePatch Method 2851
CreateProject Method 2852
DeleteAdvancedFile Method 2853
DeleteComponent Method 2854
DeleteCustomAction Method 2855
DeleteMergeModule Method 2855
DeletePathVariable Method 2856
DeleteProductConfig Method 2857
DeleteProperty Method 2857
DeleteSetupFile Method 2858
DeleteSetupType Method 2859
DeleteSQLConnection Method 2860
DeselectLanguage Method 2860
ExportProject Method 2861
ExportStrings Method 2862
ForceUpgrade Method (Advanced UI and Suite/Advanced UI) 2863
GenerateGUID Method 2864
ImportProject Method 2864
ImportStrings Method 2865
OpenProject Method 2866
RemoveFeature Method 2868
SaveProject Method 2869
ISWiAdvancedFile Object 2869
ISWiAutomaticUpgradeEntry Object 2870
Delete Method 2871
ISWiCustomAction Object 2871
ISWiComponent Object 2872
AddComponentSubFolder Method 2889
AddDynamicFileLinking Method 2890
AddEnvironmentVar Method 2891
AddFile Method 2891
AddRemoveFile Method 2892
DeviceDriverFlagsEx Property 2892
DeviceDriverSetRedist Property 2894
ImportINIFile Method 2895
ImportRegFile Method 2896
ISWiRemoveFiles Method 2897
RemoveComponentSubFolder Method 2898
RemoveDynamicFileLinking Method 2899
RemoveEnvironmentVar Method 2899
RemoveFile Method 2900

RemoveRemoveFile Method *2901*
ISWiComponentSubFolder Object 2901
DeleteFile Method *2903*
ISWiCondition Object 2903
ISWiDynamicFileLinking Object 2904
ISWiEnvironmentVar Object 2906
ISWiFeature Object 2908
AddCondition Method *2914*
AddMergeModule Method *2915*
AddObject Method *2915*
AttachComponent Method *2916*
DeleteCondition Method *2917*
RemoveComponent Method *2917*
RemoveMergeModule Method *2918*
RemoveObject Method *2919*
ISWiFile Object 2919
ISWiFolder Object 2925
AddShortcut Method *2926*
AddSubFolder Method *2927*
DeleteShortcut Method *2928*
DeleteSubFolder Method *2929*
ISWiLanguage Object 2929
AddStringEntry Method *2930*
DeleteStringEntry Method *2931*
ISWiObject Object 2931
ISWiPathVariable Object 2932
ISWiProductConfig Object 2935
AddRelease Method *2938*
DeleteRelease Method *2939*
ISWiProperty Object 2940
ISWiRelease Object 2941
Build Method *2982*
BuildTablesOnly Method *2983*
BuildTablesRefreshFiles Method *2983*
CubFile Property *2984*
SignMedia Property *2984*
ISWiRemoveFile Object 2985
ISWiSequenceRecord Object 2986
ISWiSetupFile Object 2987
ISWiSetupType Object 2988
ISWiShellProperty Object 2989
ISWiShortcut Object 2990
AddShellProperty Method *2995*
DeleteShellProperty Method *2996*
ISWiSISProperty Object 2997
ISWiSQLConnection Object 2998

AddSQLRequirement Method *3000*
AddSQLScript Method *3001*
AddSQLScriptEx Method *3001*
DeleteSQLScript Method *3002*
GetDatabaseServer Method *3003*
InsertSQLScript Method *3003*
RemoveSQLRequirement Method *3004*
ISWiSQLDatabaseServer Object 3004
ISWiSQLReplace Object 3006
ISWiSQLRequirement Object 3007
SetPredefinedRequirement Method *3008*
ISWiSQLScript Object 3009
AddSQLScriptError Method *3011*
AddSQLScriptReplacement Method *3012*
DeleteSQLReplace Method *3012*
DeleteSQLScriptError Method *3013*
ISWiSQLScriptError Object 3013
ISWiStringEntry Object 3014
ISWiUpgradeTableEntry Object 3015
Delete Method *3016*
Automation Collections 3016
ISWiAdvancedFiles Collection 3016
ISWiAutomaticUpgradeEntries Collection 3017
ISWiComponents Collection 3018
ISWiComponentSubFolders Collection 3020
ISWiConditions Collection 3021
ISWiCustomActions Collection 3022
ISWiDynamicFileLinkings Collection 3023
ISWiEnvironmentVars Collection 3024
ISWiFeatures Collection 3025
ISWiFiles Collection 3026
ISWiFolders Collection 3027
ISWiLanguages Collection 3028
ISWiObjects Collection 3029
ISWiPathVariables Collection 3030
ISWiProductConfigs Collection 3031
ISWiProperties Collection 3033
ISWiReleases Collection 3034
ISWiRemoveFiles Collection 3035
ISWiSequence Collection 3036
InsertCustomAction Method *3038*
RemoveSequenceRecord Method *3038*
ISWiSetupFiles Collection 3039
ISWiSetupTypes Collection 3040
ISWiShellProperties Collection 3041
ISWiShortcuts Collection 3042

ISWiSISProperties Collection . . . 3043
ISWiSQLConnections Collection . . . 3044
ISWiSQLDatabaseServers Collection . . . 3045
ISWiSQLReplaces Collection . . . 3046
ISWiSQLRequirements Collection . . . 3047
ISWiSQLScriptErrors Collection . . . 3048
ISWiSQLScripts Collection . . . 3049
ISWiStringEntries Collection . . . 3050
ISWiUpgradeTableEntries Collection . . . 3050
Automation Objects for Advanced UI and Suite/Advanced UI Projects . . . 3053
ISWiProject Object (Advanced UI and Suite/Advanced UI) . . . 3053
AddExitCondition Method (Advanced UI and Suite/Advanced UI) . . . 3060
AddLanguage Method (Advanced UI and Suite/Advanced UI) . . . 3061
AddPathVariable Method (Advanced UI and Suite/Advanced UI) . . . 3062
AddProperty Method (Advanced UI and Suite/Advanced UI) . . . 3062
AddSetupFile Method (Advanced UI and Suite/Advanced UI) . . . 3063
AddSuiteAction Method (Advanced UI and Suite/Advanced UI) . . . 3064
AddSuiteFeature Method (Advanced UI and Suite/Advanced UI) . . . 3065
AddSuitePackage Method (Advanced UI and Suite/Advanced UI) . . . 3066
AddSuiteRelease Method (Advanced UI and Suite/Advanced UI) . . . 3069
AddSuiteTransaction Method (Advanced UI and Suite/Advanced UI) . . . 3069
CloseProject Method (Advanced UI and Suite/Advanced UI) . . . 3070
CreateProject Method (Advanced UI and Suite/Advanced UI) . . . 3070
DeleteExitCondition Method (Advanced UI and Suite/Advanced UI) . . . 3071
DeleteLanguage Method (Advanced UI and Suite/Advanced UI) . . . 3072
DeletePathVariable Method (Advanced UI and Suite/Advanced UI) . . . 3072
DeleteProperty Method (Advanced UI and Suite/Advanced UI) . . . 3073
DeleteSetupFile Method (Advanced UI and Suite/Advanced UI) . . . 3074
DeleteSuiteAction Method (Advanced UI and Suite/Advanced UI) . . . 3075
DeleteSuiteFeature Method (Advanced UI and Suite/Advanced UI) . . . 3076
DeleteSuitePackage Method (Advanced UI and Suite/Advanced UI) . . . 3076
DeleteSuiteRelease Method (Advanced UI and Suite/Advanced UI) . . . 3077
ForceUpgrade Method (Advanced UI and Suite/Advanced UI) . . . 3078
GenerateGUID Method (Advanced UI and Suite/Advanced UI) . . . 3078
ISWiSuiteEvent Method (Advanced UI and Suite/Advanced UI) . . . 3079
OpenProject Method (Advanced UI and Suite/Advanced UI) . . . 3081
SaveProject Method (Advanced UI and Suite/Advanced UI) . . . 3082
ISWiLanguage Object (Advanced UI and Suite/Advanced UI) . . . 3083
AddStringEntry Method (Advanced UI and Suite/Advanced UI) . . . 3084
DeleteStringEntry Method (Advanced UI and Suite/Advanced UI) . . . 3085
ISWiPathVariable Object (Advanced UI and Suite/Advanced UI) . . . 3085
ISWiProperty Object (Advanced UI and Suite/Advanced UI) . . . 3089
ISWiSetupFile Object (Advanced UI and Suite/Advanced UI) . . . 3090
ISWiSuiteAction Object (Advanced UI and Suite/Advanced UI) . . . 3091
ISWiSuiteActionRef Object (Advanced UI and Suite/Advanced UI) . . . 3098
MoveTo Method (Advanced UI and Suite/Advanced UI) . . . 3099

ISWiSuiteCondition Object (Advanced UI and Suite/Advanced UI) 3100
AddCondition Method (Advanced UI and Suite/Advanced UI) *3102*
AddExtensionCondition Method (Advanced UI and Suite/Advanced UI) *3105*
AddGroup Method (Advanced UI and Suite/Advanced UI) *3106*
AddParameter Method (Advanced UI and Suite/Advanced UI) *3107*
ChangeGroupType Method (Advanced UI and Suite/Advanced UI) *3108*
DeleteCondition Method (Advanced UI and Suite/Advanced UI) *3109*
DeleteParameter Method (Advanced UI and Suite/Advanced UI) *3110*
Move Method (Advanced UI and Suite/Advanced UI) *3110*
ISWiSuiteConditionParameter Object (Advanced UI and Suite/Advanced UI) 3111
ISWiSuiteDFLFilter Object (Advanced UI and Suite/Advanced UI) 3112
CanMove Method (Advanced UI and Suite/Advanced UI) *3113*
MoveDown Method (Advanced UI and Suite/Advanced UI) *3114*
MoveUp Method (Advanced UI and Suite/Advanced UI) *3115*
ISWiSuiteDynamicFileLink Object (Advanced UI and Suite/Advanced UI) 3115
AddFilter Method (Advanced UI and Suite/Advanced UI) *3116*
DeleteFilter Method (Advanced UI and Suite/Advanced UI) *3117*
ISWiSuiteEvent Object (Advanced UI and Suite/Advanced UI) 3118
AddSuiteActionRef Method (Advanced UI and Suite/Advanced UI) *3118*
DeleteSuiteActionRef Method (Advanced UI and Suite/Advanced UI) *3119*
ISWiSuiteExitCondition Object (Advanced UI and Suite/Advanced UI) 3120
ISWiSuiteFeature Object (Advanced UI and Suite/Advanced UI) 3121
AddCondition Method (Advanced UI and Suite/Advanced UI) *3124*
AddSuiteSubFeature Method (Advanced UI and Suite/Advanced UI) *3124*
AttachPackage Method (Advanced UI and Suite/Advanced UI) *3125*
DeleteCondition Method (Advanced UI and Suite/Advanced UI) *3126*
DeleteSuiteSubFeature Method (Advanced UI and Suite/Advanced UI) *3126*
RemovePackage Method (Advanced UI and Suite/Advanced UI) *3127*
ISWiSuiteFile Object (Advanced UI and Suite/Advanced UI) 3128
ISWiSuiteFolder Object (Advanced UI and Suite/Advanced UI) 3129
AddDynamicFileLink Method (Advanced UI and Suite/Advanced UI) *3130*
AddFile Method (Advanced UI and Suite/Advanced UI) *3131*
AddFolder Method (Advanced UI and Suite/Advanced UI) *3132*
DeleteDynamicFileLink Method (Advanced UI and Suite/Advanced UI) *3132*
DeleteFile Method (Advanced UI and Suite/Advanced UI) *3133*
DeleteFolder Method (Advanced UI and Suite/Advanced UI) *3134*
ISWiSuiteOperation Object (Advanced UI and Suite/Advanced UI) 3134
ISWiSuitePackage Object (Advanced UI and Suite/Advanced UI) 3140
AddDetectCondition Method (Advanced UI and Suite/Advanced UI) *3159*
AddEligibleCondition Method (Advanced UI and Suite/Advanced UI) *3159*
AddSuitePackage Method (Advanced UI and Suite/Advanced UI) *3160*
DeleteCondition Method (Advanced UI and Suite/Advanced UI) *3161*
MoveDown Method (Advanced UI and Suite/Advanced UI) *3162*
MoveUp Method (Advanced UI and Suite/Advanced UI) *3163*
ISWiSuiteRelease Object (Advanced UI and Suite/Advanced UI) 3163
Build Method (Advanced UI and Suite/Advanced UI) *3175*

Automation Collections for Advanced UI and Suite/Advanced UI Projects . . . 3176
ISWiLanguages Collection . . . 3176
ISWiPathVariables Collection . . . 3177
ISWiProperties Collection . . . 3178
ISWiSetupFiles Collection . . . 3179
ISWiSuiteActionRefs Collection . . . 3179
ISWiSuiteActions Collection . . . 3180
ISWiSuiteConditionParameters Collection . . . 3180
ISWiSuiteConditions Collection . . . 3181
ISWiSuiteDFLFilters Collection . . . 3182
ISWiSuiteDynamicFileLinks Collection . . . 3183
ISWiSuiteEvents Collection . . . 3183
ISWiSuiteExitConditions Collection . . . 3184
ISWiSuiteFeatures Collection . . . 3185
ISWiSuiteFiles Collection . . . 3186
ISWiSuiteFolders Collection . . . 3187
ISWiSuitePackages Collection . . . 3188
ISWiSuiteReleases Collection . . . 3188
Object Reference for Expressions in Advanced UI and Suite/Advanced UI Projects . . . 3191
Feature Object . . . 3191
File Object . . . 3193
Parcel Object . . . 3195
Platform Object . . . 3197
Registry Object . . . 3199
InstallShield Custom Action Reference . . . 3201
Command-Line Tools . . . 3213
Compile.exe . . . 3214
ISCmdBld.exe . . . 3220
ISBuild.exe . . . 3228
IISscan.exe . . . 3229
iSign.exe . . . 3230
ReleasePackager.exe . . . 3231
SetupIni.exe . . . 3233
MsiExec.exe Command-Line Parameters . . . 3233
Setup.exe and Update.exe Command-Line Parameters . . . 3238
Advanced UI and Suite/Advanced UI Setup.exe Command-Line Parameters . . . 3252
Setup.exe (InstallScript Projects) . . . 3256

13 Frequently Asked Questions . . . 3259

Glossary . . . 3263

Index . . . 3293

1

InstallShield 2016 Help Library

You can use InstallShield to rapidly build, test, and deploy installations that target Windows-based systems.

The InstallShield Help Library contains information about the functionality and features of InstallShield. The Help Library contains the following sections:

Table 1-1 • Help Library Sections

Section	Description
What's New in InstallShield 2016	Informs you about the changes in InstallShield 2016.
What Was New in Earlier Versions of InstallShield	Informs you about changes that were made in earlier versions of InstallShield.
Target System Requirements	Lists the requirements for target systems.
Launching InstallShield with vs. Without Administrative Privileges	Alerts you to functionality that is not available if you are running InstallShield without administrative privileges; also describes a potential problem that may occur if you switch between full Administrator and non-Administrator contexts and you use mapped-drive locations in your projects.
Developing and Building Installations on 32-Bit vs. 64-Bit Systems	Alerts you to differences that you may notice if you use InstallShield on some 32-bit systems compared to some 64-bit systems.
Using Help	Provides information about the InstallShield documentation.
Getting Started	Contains information to help you become familiar with InstallShield, begin creating an installation project, and customize the InstallShield user interface.
Tutorials	Leads you step-by-step through the process of creating InstallScript and Basic MSI installation projects, and creating global installations.

Table 1-1 • Help Library Sections (cont.)

Section	Description
Creating Installations	Explains how to create user-friendly, reliable installations and guides you through every step of the process—from specifying information for Add or Remove Programs to building, testing, and deploying an installation.
Creating Advanced UI and Suite/Advanced UI Installations	Contains an overview, plus some detailed how-to information, about Suite/Advanced UI and Advanced UI projects. The Suite/Advanced UI project type that is available in the Premier edition of InstallShield lets you package multiple .msi packages, .msp packages, InstallScript packages, .exe packages, sideloading UWP app packages (.appx), and Windows Installer transactions as a single installation while providing a contemporary, customizable user interface. The Advanced UI project type that is available in the Professional edition of InstallShield lets you provide a contemporary, customizable user interface for a single .msi package, .msp package, or InstallScript package.
Designing InstallShield Prerequisites and Other Redistributables	Introduces some basic concepts to help you get started designing your own InstallShield prerequisites, merge modules, and InstallScript objects that can be used in any of your installation projects or distributed for use by other installation developers.
Modularizing Installation Projects to Distribute Development Work and Enable Reuse	Introduces developer installation manifests (DIMs), a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete portion of a product installation.
Updating Applications	Leads you through steps for planning and implementing the various types of upgrades and patches for updating a product. Also explains how you can use FlexNet Connect to notify end users about upgrades and patches that are available.
Creating Customized Virtual Applications	Explains how to use InstallShield to create customized virtual applications.
Additional Installation Options	Discusses a broad range of options available in InstallShield: creating multilingual installations, installing multiple instances of a product, building conditional statements, searching for installed data, editing installation tables, and more.
Integrating InstallShield with External Applications	Contains details about integrating InstallShield with third-party tools such as source code control software, Microsoft Visual Studio, and Microsoft Visual Studio Team Foundation Server (TFS).
Automating Installation Development and Build Processes	Provides information about the InstallShield automation interface, which enables you to automate processes for creating installation projects without having to directly open the InstallShield user interface.

Table 1-1 • Help Library Sections (cont.)

Section	Description
Reference	Contains comprehensive reference information for the InstallShield user interface; the InstallScript language; errors and warnings that might occur when you create, compile, build, or run your installation; tools that you can use from the command line to perform tasks such as building a release and running an installation; the objects that you can use to embed object expressions in various settings of an Advanced UI or Suite/Advanced UI project to search target systems; the InstallShield custom actions that are added to projects; and objects, methods, properties, and collections used to modify an installation project through the automation interface.
Frequently Asked Questions	Directs you to help topics that answer many commonly asked questions about InstallShield and project creation.
Glossary	Contains a collection of terms and their meanings.



Note • *Because the InstallShield Help Library is designed to interact with InstallShield, it is recommended that you open the help from within InstallShield. Copying the help files to another folder or system causes many of its features to work incorrectly.*

For answers to many commonly asked questions and new information about InstallShield that do not appear in the documentation, visit the Knowledge Base.

What's New in InstallShield 2016

For information about new features added in InstallShield 2016, refer to the following sections:

- New Features in InstallShield 2016 SP2
- New Features in InstallShield 2016 SP1
- New Features in InstallShield 2016

New Features in InstallShield 2016 SP2

Integration with FlexNet Code Aware

InstallShield now includes integration with FlexNet Code Aware, an automated open source risk assessment and package discovery solution that enables you to quickly scan your products for security and intellectual property (IP) compliance risk.

The current release of FlexNet Code Aware supports analysis of the following files:

- Java Packages

- Node Packages
- Nuget Packages
- RPM Packages
- Ruby Packages
- EXE & DLL Files

Security vulnerabilities are looked up against the National Vulnerability Database (NVD).

Running FlexNet Code Aware

FlexNet Code requires a separate license from InstallShield. There is also trial/evaluation version. For more information, refer to the FlexNet Code Aware product page of the Flexera Software website.

To run FlexNet Code Aware from within InstallShield, click **Scan Project using FlexNet Code Aware** from the InstallShield **Project** menu. This menu option is disabled out if you are not currently in an open InstallShield project.

A FlexNet Code Aware icon  is also available on the InstallShield standard toolbar.

When FlexNet Code Aware completes the scan of your project, a summary displays showing the number of files scanned, and the number of open-source packages and vulnerabilities found. A **View report** button is provided if you have a fully licensed version of FlexNet Code Aware. For more information about the details provided in this report, refer to Reading the FlexNet Code Aware Report.



Reading the FlexNet Code Aware Report

Note *• The FlexNet Code Aware Report is not available in trial/evaluation mode. A fully licensed version of FlexNet Code Aware is required.*

To view the FlexNet Code Aware Report, click **View report** on the summary dialog that appears after FlexNet Code Aware has scanned your project.

The FlexNet Code Aware report consists of several sections:

- The initial Summary View presents the user with a **Scan Summary**, **Operational Risk** assessment, **Security Vulnerability Exposure,** and **License Exposure**.
  - The **Scan Summary** section provides details regarding the codebase that was scanned, including a breakdown of file types, percent of files analyzed, and number of findings.
  - The **Operational Risk** section provides a composite risk rating based on the combination of packages with Intellectual Property (IP) issues and packages with Security Vulnerabilities.
  - The **Security Vulnerability Exposure** and **License Exposure** sections provide a breakdown of the types and categories of identified issues.
- The Package Inventory View, available by clicking **view full package inventory** in the **Scan Summary** section, provides a complete list of discovered open source and third-party packages with associated licenses, security vulnerabilities, dependencies, and detected copyright statements.

The Package Inventory View provides filters that you can use to execute targeted queries to refine the list to various package types of interest.

The following figures show the initial Summary View of a sample FlexNet Code Aware Report.

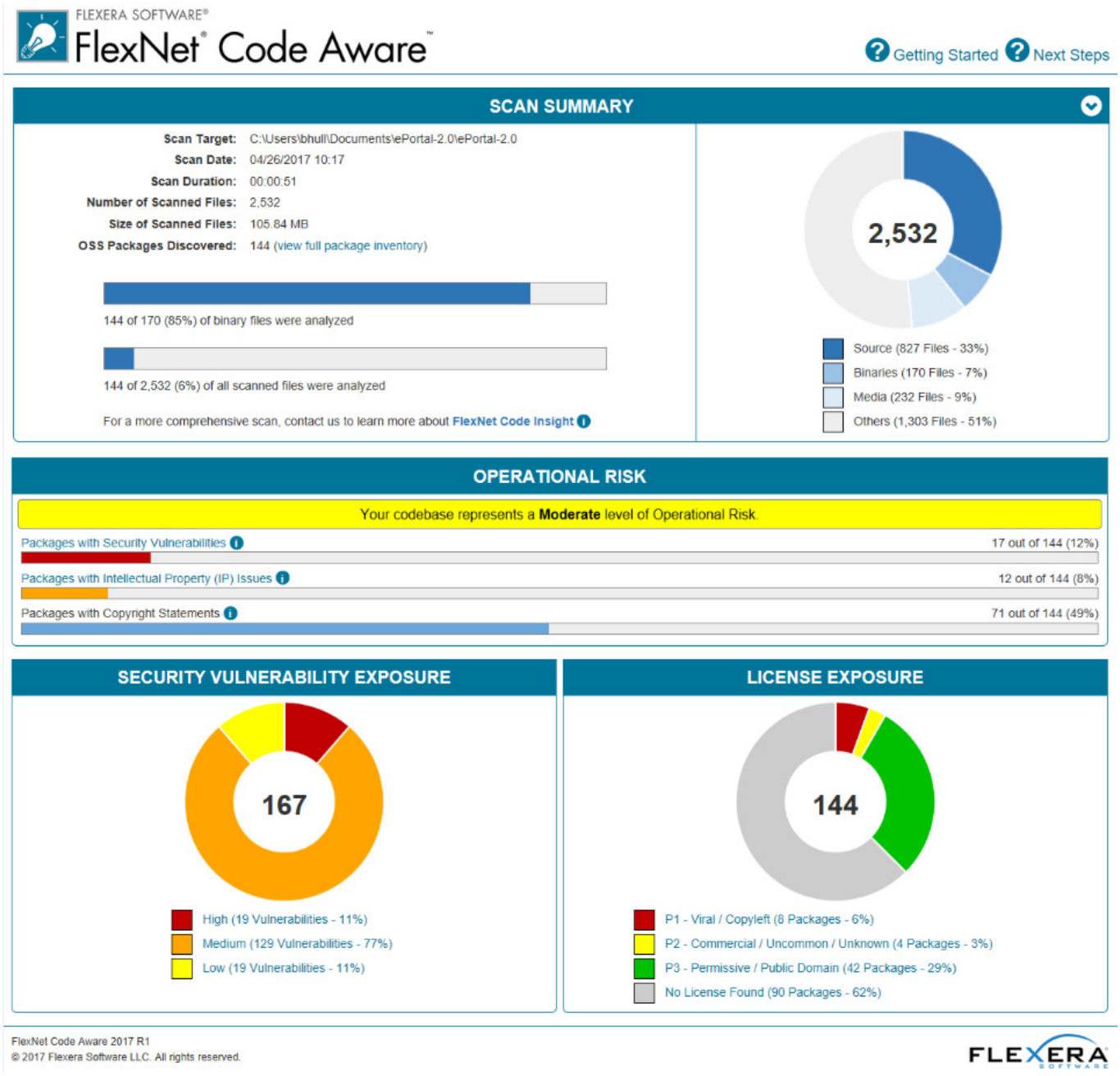


Figure 1-1: FlexNet Code Aware Initial Summary View

The following figures show the Package Inventory View of a sample FlexNet Code Aware Report.

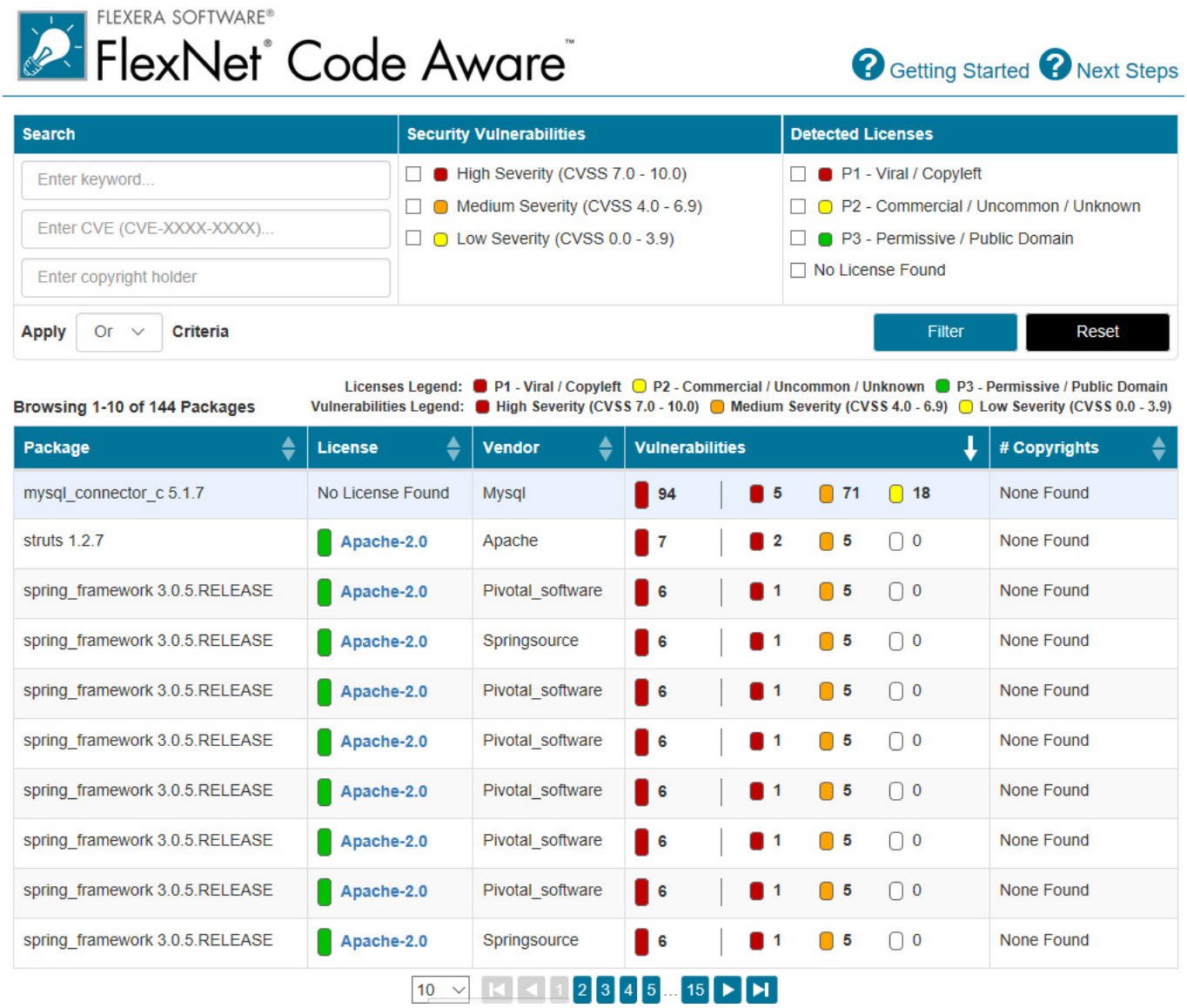
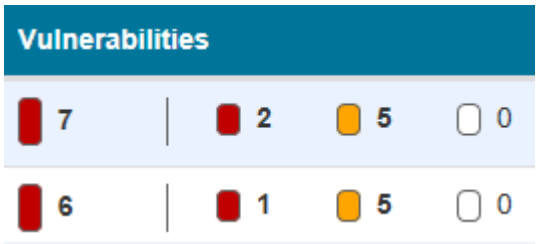


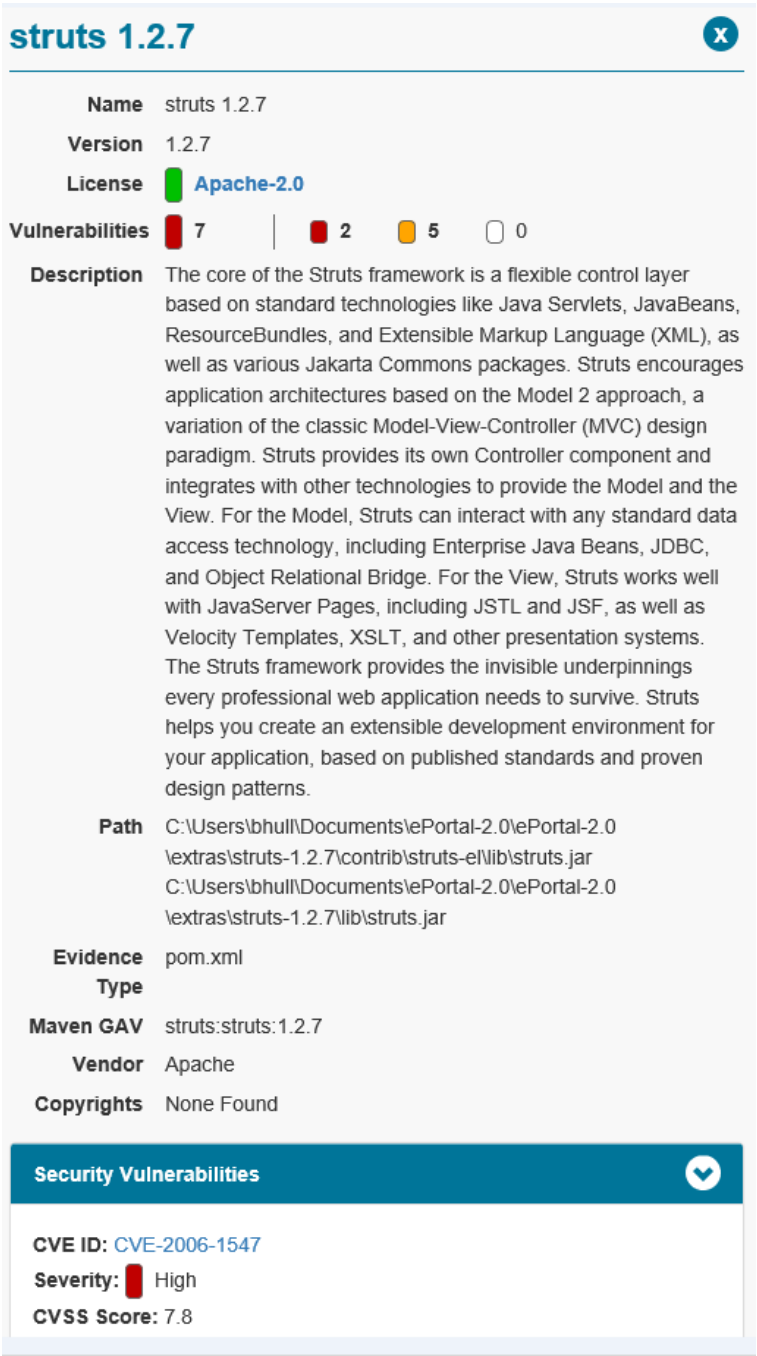
Figure 1-2: FlexNet Code Aware Package Inventory View

Viewing Package Details

Click a vulnerability count listed in the Vulnerabilities column of the Package Inventory report page for each package you want to review:



The **Vulnerabilities detail** page appears, covering a portion of the Package Inventory report:



New Features in InstallShield 2016 SP1

Support for Microsoft Visual Studio 2017

InstallShield includes support for Visual Studio 2017. You can create InstallShield projects from within this version of Visual Studio.

New Features in InstallShield 2016

InstallShield 2016 includes the following new features:

- Support for the Latest Releases of Windows Operating Systems
- Microsoft SQL Server 2016 Support
- Support for Creating UWP App Packages
- SQL Support Added to Suites
- Tile Configurations
- New InstallShield Prerequisites for Microsoft Visual C++ 2015, .NET Framework 4.6, and More
- Predefined System Searches for Adobe Reader, Microsoft Office and the .NET Framework

Support for the Latest Releases of Windows Operating Systems

InstallShield 2016 supports the latest releases of the Windows operating system:

- Windows 10 Anniversary Update
- Windows Server 2016

Not only can you install InstallShield on these operating systems, but you can also create installers that target these operating systems.

Microsoft SQL Server 2016 Support



Project • *Microsoft SQL Server 2016 support is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

InstallShield now includes support for running SQL scripts on SQL Server 2016 database servers. In addition, InstallShield includes SQL Server 2016 in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports.

If your installation targets SQL Server 2016, the SQLBrowse run-time dialog that is displayed when end users choose to browse for a database server can now list instances of SQL Server 2016, SQL Server 2016 Express, and SQL Server 2014 Express LocalDB. In addition, the SQLBrowse run-time dialog that is displayed when end users choose to browse for a database catalog can now list catalogs on the specified SQL Server 2016 database server.

See New InstallShield Prerequisites for Microsoft Visual C++ 2015, .NET Framework 4.6, and More for a complete list of new InstallShield prerequisites added to InstallShield.



Note • *Microsoft SQL Server 2016 is now 64-bit (x64) only.*

Support for Creating UWP App Packages

Project • *UWP app creation is available in Basic MSI projects.*

Important • *The Windows 10 Anniversary Update is required for installing and testing a UWP app package (.appx) with desktop extensions (Desktop Bridge). To digitally sign the UWP app package, InstallShield must be installed on a Windows 10 machine or a machine with the Windows 10 SDK installed.*

The UWP app package (.appx) format is the simple and secure packaging format used to distribute and install apps on Windows 8.x and 10 and is the only format allowed for Universal Windows Platform (UWP) apps. Benefits of UWP app packages include:

- High availability, reliability, and durability, resulting in applications that operate continuously without failure for extended periods of time
- A smooth installation experience through static builds that require minimal configuration and no customizable UI
- The option to sell or provide the application through the Windows Store
- The ability to leverage UWP functionality such as live tiles as well as the ability to utilize UWP APIs
- The only package format with native support on Windows Nano Server

InstallShield now supports creating the UWP app package format (.appx) and its desktop and server extensions through an alternate build output and provides suitability testing to help you identify items unsuitable for the UWP app package format. Refer to the following subsections for details about new functionality added to InstallShield to support the creation of UWP app packages.

- UWP App Settings in the Releases View
- UWP App Logo Customization in the Shortcuts View
- UWP App Suitability Testing
- New UWP Condition Checks added to Suites

UWP App Settings in the Releases View

When you select a release in the Releases view, a new per-release tab titled **Windows App** has been added that includes settings to create a UWP app package. Here, various core settings can be specified that impact the UWP app package build process. In particular, the **Distribution Method** and **Include Desktop Extensions** or **Include Server Extensions** options will affect what warnings or errors are encountered for certain kinds of installer project data.

For complete information on these new settings, see Windows App Tab for a Release.

UWP App Logo Customization in the Shortcuts View

In addition to the new UWP app settings added to the Releases view, there are new settings to configure the tiles created in an UWP app package. These settings are located in the new **UWP App Package Tile Overrides** area of the Shortcuts view.

For complete information on these new settings, see Shortcut Settings.

UWP App Suitability Testing

InstallShield has added the new InstallShield UWP App Suitability Suite that scans an .msi package for signs of items that are unsuitable for the UWP app package format. To access it, on the **Build** menu, point to **Validation**, and then click **InstallShield UWP App Suitability Suite**.

The InstallShield UWP App Suitability Suite provides a report in the Releases view that indicates all tests that found issues and for each issue, an associated column in the report indicates applicability to the known UWP app variants. For traditional CUBs, these columns are not populated. You can view this report by navigating to the Releases view and selecting the Validations folder under your release. For complete information, including descriptions of the new ISUWP validations included, see InstallShield UWP App Suitability Suite.

New UWP Condition Checks added to Suites

When you are building a conditional statement for an exit, detection, eligibility, feature, or wizard interface condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can select from a number of different types of checks that you want to be evaluated on target systems. The following condition checks have been added to Suites.

- **UWP App Package Eligible**—Check target systems for the run time dependencies of the UWP app package to prevent any attempts to install a UWP app package onto a version of Windows or Windows Server that does not support it.

  Note *• This condition is available only for the eligibility condition of a UWP app package. If it is used in any other package type, it will not function correctly.*

- **UWP Type Present**—Check target systems for the presence of UWP functionality. For example, to create a conditional statement that checks for the presence of the Desktop Bridge, check for the type `Windows.ApplicationModel.FullTrustProcessLauncher`. This can be used to conditionally block installation, or to choose between installing .msi and UWP app package (.appx).

  Note *• This always evaluates as false on operating systems before Windows 10. Use of the Type Name subsetting* `Windows.ApplicationModel.FullTrustProcessLauncher` *requires Windows 10 Anniversary Update or newer.*

SQL Support Added to Suites

Project *• This information applies to Suite/Advanced UI projects.*

Edition *• The Suite/Advanced UI project type is available in the Premier edition of InstallShield.*

SQL servers are integral to many applications, especially those that benefit from the multiple package support provided by InstallShield Suite installations. Previously, InstallShield SQL support was limited to Basic MSI, InstallScript, and InstallScript MSI projects. Now, SQL support has been added to Suite/Advanced UI projects, giving you the ability to:

- Add a New SQLLogin Predefined Wizard Page
- Execute SQL Statements Directly from Suites

To learn more, see the following topics:

- Adding a SQLLogin Predefined Wizard Page in a Suite/Advanced UI Project
- Configuring an Action for an Element in the Wizard Interface
- Predefined Task Pages Panel

Add a New SQLLogin Predefined Wizard Page

InstallShield adds SQL support to Suite/Advanced UI projects by providing a new **SQLLogin** predefined wizard page that can be added to your Suite/Advanced UI project. Previously, if you added an .msi package containing SQL support to a Suite/Advanced UI project, the Advanced UI or Suite/Advanced UI setup launcher automatically suppressed the user interface of Windows Installer. You were required to manually create a custom SQLLogin wizard page for your Suite/Advanced UI project installation.

When adding a new predefined page to your project, select the **Enter login information for a database server** task page and complete the panels in the wizard as needed. The SQLLogin predefined wizard page is then added to your project. This SQLLogin wizard page lets end users enter database server login information (database server name, authentication credentials, database catalog name, etc.) in order to establish a connection to the database server that is targeted by one or more .msi packages in the suite.

Adding a SQLLogin wizard page to your project allows you to:

- Specify properties that identify the SQL login settings in the Suite project and then select the .msi package that receives these properties
- Specify the properties that identify the SQL login settings in the .msi package
- Choose the database technology (Microsoft SQL Server, Microsoft Windows Azure, MySQL, or Oracle) and select the ODBC driver to be targeted

Execute SQL Statements Directly from Suites

Suite/Advanced UI projects now support directly executing SQL statements on SQL database servers from the user interface, which helps allow SQL database servers to be examined before proceeding with the installation.

The SQL query result can then be accessed in a Suite property. To provide this support, a **Run a SQL String** option is now available in the New Action menu of UI events. The SQL statement is executed using properties and database metadata specified by additional new options available in the New Action menu of UI events: **Configure Database Metadata** and **Override SQL Login Properties**.

Tile Configurations



Project • *This information applies to Basic MSI, InstallScript MSI, and InstallScript project types.*

Windows 8 introduced a grid of application tiles to the Start screen, replacing the usual list of shortcuts, and also presented tiles in place of shortcuts. InstallShield supports customizing the appearance of a desktop app's tile on the Start screen. The following tile configuration settings are available:

- A toggle between light or dark text when including the app name on medium-sized (150x150) tiles
- Choice of tile background color
- Option to use custom tile images (small: 70x70 and medium:150x150)

- Preference to show or hide the app name on medium-sized tiles

The **Tile Configurations** node appears in the main **Shortcuts** view and in each component's **Shortcuts** subview. Any applicable tile configurations are listed.

To learn more, see the following topics:

- Configuring the Appearance of a Desktop App's Tile on the Start Screen
- Tile Configuration Settings

New InstallShield Prerequisites for Microsoft Visual C++ 2015, .NET Framework 4.6, and More



Project *• InstallShield prerequisites can be added to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects.*

InstallShield includes the following new InstallShield prerequisites:

- Microsoft .NET Framework 4.6.1 Full
- Microsoft .NET Framework 4.6.1 Web
- Microsoft ReportViewer 2015
- Microsoft SQL Server 2014 Express System CLR Types (x86)
- Microsoft SQL Server 2016 Express RTM (x64)
- Microsoft SQL Server 2016 Express RTM LocalDB (x64)
- Microsoft Visual C++ 2015 Update 3 Redistributable Package (x86)
- Microsoft Visual C++ 2015 Update 3 Redistributable Package (x64)
- Windows Management Framework 4.0 for Windows 7 SP1 and Server 2008 R2 SP1 (x64)
- Windows Management Framework 4.0 for Windows Server 2012 (x64)
- Windows Management Framework 5.0 for Windows 7 SP1 (x86)
- Windows Management Framework 5.0 for Windows 7 SP1 and Server 2008 R2 SP1 (x64)
- Windows Management Framework 5.0 for Windows 8.1 (x86)
- Windows Management Framework 5.0 for Windows 8.1 and Server 2012 R2 (x64)
- Windows Management Framework 5.0 for Windows Server 2012 (x64)

These prerequisites install the appropriate technologies on supported target systems.



Note *• The* **Web** *prerequisite for the .NET Framework requires an Internet connection. This prerequisite downloads the required redistributable files if appropriate. The* **Full** *prerequisite for the .NET Framework is a stand-alone installation that does not require an Internet connection.*

Predefined System Searches for Adobe Reader, Microsoft Office and the .NET Framework



Project *• Predefined System Searches apply to Basic MSI and InstallScript MSI projects.*

InstallShield has new predefined system searches:

- Adobe Reader 11
- Adobe Reader DC
- Microsoft Office 2013
- Microsoft Office 2016
- Microsoft .NET Framework 4.5.1
- Microsoft .NET Framework 4.5.2
- Microsoft .NET Framework 4.6
- Microsoft .NET Framework 4.6.1

If your installation requires one or both of these, you can use the System Search view or the Installation Requirements page in the Project Assistant to add these system searches to your project. When end users launch your installation, Windows Installer checks the target system to see if the requirements are met; if they are not met, the installation displays the error message that is defined for the system search.

Enhancements

InstallShield 2016 includes the following new enhancements:

- Direct Editor View Enhancements
- Suite UI Enhancements
- Kill Process Custom Action Enhanced
- Ability to Set the Default Value Used for Component Attributes
- Ability to Filter Items by Features in Additional Views
- Digital Signature Updates

Direct Editor View Enhancements



Project *• The Direct Editor is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*

- *MSM Database*
- *QuickPatch*
- *Transform*

InstallShield now adds several Direct Editor enhancements that provide visual insights into tables, schema information, and validation errors designed to boost the productivity of setup authors or packagers who use the Direct Editor for troubleshooting to identify and resolve advanced problems. These enhancements are described in the following sections:

- Directory Table Displays Resolved Target Directory Path
- Column Header Schema Information Tooltips
- Table Record Reference Tracking
- Broken References Indicator

For complete information, see Direct Editor.

Directory Table Displays Resolved Target Directory Path

When showing the Directory table, InstallShield shows a read-only, grayed out column that displays the resolved path of each row's directory location. This column is not actually stored in the project file. You can sort by its visible text, but you cannot insert, update, or delete its values.

Column Header Schema Information Tooltips

InstallShield now provides tooltips added to column headers that display the following schema information to indicate the column data type allowed:

- **nullable**—The column can be left empty.
- **required**—The column must contain a non-empty value.
- **char(*nn*)**—Character string with fixed-length *nn*.
- **small integer**—Integer numerical (no decimal), containing a value from -32767 to +32767.
- **long integer**—Integer numerical (no decimal), containing a value from -2147483647 to +2147483647.
- **localizable**—This column contains a string that can be translated. Columns without this marker are not localized.
- **stream—**Binary stream, such as the contents of a file.



***Tip** • The **Directory**, **Binary**, and **CustomAction** Direct Editor tables display several of these column types.*

Table Record Reference Tracking

The Direct Editor now includes a Reference Tracking pane that lets you easily visualize relationships between table records. A **Show Reference Tracking** button is added to at the top of the Direct Editor that provides a show/hide toggle of the pane.

Each record may reference one or more other records or be referenced by one or more other records. When a record is highlighted that refers to other records or is referred from other records, the Reference Tracking pane is populated with a **Reference Tables** section showing the tables in which the references reside and an additional section that displays the actual record references. The record references section includes arrow icons that indicate the direction of the reference, where:

- A green arrow pointing to the right indicates that the record that is selected in the Direct Editor table **references** the record displayed in the Reference Tracking pane.
- A blue arrow pointing to the left indicates that the record selected in the Direct Editor table is **referenced by** the record displayed in the Reference Tracking pane.
- Two arrows in both directions (a green arrow pointing right and a blue arrow pointing left) that indicate the record selected in the Direct Editor table **references** *and is* **referenced by** the record displayed in the Reference Tracking pane.



***Note** • When multiple Direct Editor records are selected, only the "focused" record's references are shown. In addition, if multiple tables appear in the **Reference Tables** section, this indicates that the record selected in the Direct Editor table references or is referenced by records in multiple tables. You can click any table in the **Reference Tables** section to view the associated records references.*



***Tip** • In the Reference Tracking pane, you can quickly jump between record references by double-clicking within a cell.*

Broken References Indicator

There are instances where a Direct Editor table record might reference a foreign key record that no longer exists. InstallShield now displays the cells of such broken references with a red fill color to call attention to the broken reference.

For example, if the **Directory_** column in the in the **Component** table references a directory name that is not found in the **Directory** table, then the **Directory_** column is filled in red.



***Note** • The Direct Editor broken reference indicator is unrelated to the **Maintain referential integrity** check box on the Preferences tab of the Options dialog box. While the **Maintain referential integrity** setting is designed to update the foreign keys when you modify a primary key, the purpose of the broken reference indicator is to display broken references to help you easily identify orphaned records. Therefore, broken references are displayed regardless of the selection of the **Maintain referential integrity** setting.*

Suite UI Enhancements



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

To better support various use cases, InstallShield has added the following functionality to Advanced UI and Suite/Advanced UI projects:

- New Close Window and Stop Event UI Actions
- Suite Loading Splash Screen Added

New Close Window and Stop Event UI Actions

InstallShield includes the following new UI actions:

Table 1-2 • New Control Actions

Type of Action	Description
Close Window	This type of action closes a main wizard page or secondary window, or in some cases, provides conditional closing of a secondary window. Close Window accepts parameters corresponding to the following predefined Return Code IDs: IDOK, IDCANCEL, IDABORT, IDRETRY, IDIGNORE, IDYES, IDNO, and IDCLOSE. The behavior of the Close Window action differs slightly on wizard pages and secondary windows, as noted following: • For wizard pages, the Close Window action prompts the end user to cancel if its Return Code parameter is set to IDCANCEL (and then interrupts the wizard if the end user specifies Yes). For all other Return Code IDs, the wizard immediately closes. • For secondary windows, the Close Window action closes the secondary window and in special cases such as with the ISRMFilesInUse and ISRMFileInUse secondary windows, the specified Return Code value is returned. InstallShield currently provides the following secondary windows that contain customized behavior that is dependent upon the specified Return Code ID: • ISDownloadProgress • ISPromptForSourceMedia • ISFilesInUse • ISRMFilesInUse • ISUpgradeParcel • ISSQLBrowse

Table 1-2 • New Control Actions (cont.)

Type of Action	Description
Stop Event	This type of action allows you to conditionally stop further actions from being processed. For example, this action can be used to prevent the default behavior of a button from occurring.

To learn more, see Configuring an Action for an Element in the Wizard Interface.

Suite Loading Splash Screen Added

During a suite installation where the loading process takes more than half a second, InstallShield will now display a splash screen to indicate that the program has launched and that a loading process needs to complete before the Install Welcome dialog appears. For the splash screen, InstallShield utilizes the largest provided version of the **setup.exe** icon and includes a progress bar on it.

Kill Process Custom Action Enhanced



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

A new **Processes** setting has been added to the Kill-Process Custom Action settings that lets you directly enter executable file names or PIDs of the processes that you want to terminate without having to create a property using the Property Manager and format its value correctly for the action to work.

To learn more, see Calling a Kill-Process Custom Action.



Tip • *The value of the **Processes** setting may be written to the **ISTerminateProcesses** property. If you have additional kill-process custom actions that do not specify a value in the **Processes** setting, such as those migrated from InstallShield 2015 or earlier, shared use of the **ISTerminateProcesses** property may result in undesired behavior.*

Ability to Set the Default Value Used for Component Attributes

Support for a new property has been added to the **InstallShield** table that lets you set the default value used for component attributes. If a property of **MsiComponentAttributes** exists in the **InstallShield** table, its value overrides the default **Attributes** column value in the **Component** table from 8 to whatever is specified.

For example, to make new components 64-bit, add 256 to the **MsiComponentAttributes** value. You can specify 264 (for 64-bit shared) or optionally enter 256 (for 64-bit unshared). In doing so, the **64-Bit Component** setting and **Shared** settings (shown on the in the General area of the Components view) will be updated to Yes or No accordingly.

For more information about the bit values used in calculating the **Attributes** column of the Component table, refer to the Component Table page in the MSDN Library.

Note *• To set the default value used for component attributes, the* ***MsiComponentAttributes*** *property must be updated manually in the* ***InstallShield*** *table in the* ***Direct Editor*** *of each project. The* ***Template Summary*** *setting for a product configuration is ignored for this use case.*

Ability to Filter Items by Features in Additional Views

The following views now contain a View Filter that lets you filter the view list by any feature in your project:

- **Environment Variables View**—You can use the View Filter list at the top of this view to show and hide environmental variables that are associated to a particular feature in your project. You can also select a feature from the View List in order to associate only that feature with a subsequent event (e.g., the creation, modification, or removal of an environmental variable). Lastly, to see all of the environmental variables that are in your project, select the All Application Data option in the View Filter list. For more information, see Environment Variables View.
- **Text File Changes View**—You can use the View Filter list at the top of this view to show and hide text file change sets that are associated to a particular feature in your project. You can select a feature from the View List in order to associate only that feature with a subsequent event (e.g., the creation, modification, ordering, or removal of change sets). The resulting modification takes place at run time on the target system when the feature is installed. Lastly, to see all of the text file change sets that are in your project, select the All Application Data option in the View Filter list. For more information, see Text File Changes View.
- **INI File Changes View**—You can use the View Filter list at the top of this view to show and hide initialization (.ini) files that are associated to a particular feature in your project. You can select a feature from the View List in order to associate only that feature with a subsequent event (e.g., the creation, importing, modification, or removal of .ini files). The resulting modification takes place at run time on the target system when the feature is installed. Lastly, to see all of the .ini files that are in your project, select the All Application Data option in the View Filter list. For more information, see INI File Changes View.

Digital Signature Updates

Beginning with InstallShield 2015, support was added to enable you to use digital certificates that use the SHA-256 hashing algorithm for signing your installations and files at build time.

In InstallShield 2016, support for SHA-256 digital certificates has been enhanced for Windows Installer and InstallScript projects to:

- Give you the ability to specify a digest type using the new **Signature Digest** drop-down on the Certificate Selection Dialog Box
- RFC3161 timestamping is now supported and can be specified in **settings.xml,** noting that:
  - **DigitalSignature/@Timestamp** can be an **Authenticode** or **RFC3161 server** for .msi, .exe, and .dll files
  - **DigitalSignature/@TimestampRFC3161** used for UWP app package files must be an **RFC3161 server**
- Handle similarly-named certificates in the Certificate Store

Important *• Any new signatures created or timestamped after Jan 1, 2016 must be SHA-256-based signatures. Any files signed with an SHA-1 certificate need to have a timestamp showing a date and time prior to Jan 1, 2016 in order to continue to be supported. Those files will still be allowed through the 'Mark-of-the-web" system until Jan 14, 2020, when all SHA-1 support will stop in all current versions of Windows.*

What Was New in Earlier Versions of InstallShield

This section describes features and enhancements that were released in earlier versions of InstallShield:

- What's New in InstallShield 2015 SP1
- What's New in InstallShield 2015
- What's New in InstallShield 2014 SP1
- What's New in InstallShield 2014
- What's New in InstallShield 2013 SP1
- What's New in InstallShield 2013
- What's New in InstallShield 2012 Spring SP1
- What's New in InstallShield 2012 Spring
- What's New in InstallShield 2012 SP1
- What's New in InstallShield 2012
- What's New in InstallShield 2011
- What's New in InstallShield 2010 Expansion Pack for Visual Studio 2010
- What's New in InstallShield 2010 SP1
- What's New in InstallShield 2010
- What's New in InstallShield 2009 SP2
- What's New in InstallShield 2009 SP1
- What's New in InstallShield 2009
- What's New in InstallShield 2008
- What's New in InstallShield 12 SP1
- What's New in InstallShield 12

What's New in InstallShield 2015 SP1

New Features

InstallShield includes the following new features.

Support for Windows 10

InstallShield has support for Windows 10.

Support for Microsoft Visual Studio 2015

InstallShield includes support for Visual Studio 2015. You can create InstallShield projects from within this version of Visual Studio.

Support for Microsoft App-V 5.1

InstallShield and the Microsoft App-V Assistant in InstallShield include support for creating virtual applications that can run on Microsoft App-V 5.1 clients.

New InstallShield Prerequisite for App-V 5.1

InstallShield includes a new InstallShield prerequisite that can be used when including a Setup.exe for a generated App-V 5.x package. Note that a Setup.exe setup launcher is required if the InstallShield prerequisite needs to be included in the release:

- Microsoft App-V 5.1 Desktop Client

The redistributable file for this InstallShield prerequisite is not available for download from within InstallShield, since you must obtain it from Microsoft. Once you obtain the redistributable from Microsoft, place it in the location that is displayed when you are editing the prerequisite in the InstallShield Prerequisite Editor. For more information on the prerequisites necessary, see https://technet.microsoft.com/en-us/library/mt346482.aspx.

Windows 10 Added to Operating System Options in App-V Assistant for App-V 5.x Versions

App-V 5.1 introduced support for Windows 10 32-bit and 64-bit operating systems. Accordingly, on the Package Information page of the Microsoft App-V Assistant, the list of available operating systems that are displayed when App-V 5.x is selected has been updated to include:

- Windows 10 32-bit
- Windows 10 64-bit

New InstallShield Prerequisites for Microsoft Visual C++ 2015 and .NET Framework 4.6

InstallShield includes new InstallShield prerequisites that you can add to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects:

- Microsoft Visual C++ 2015 Redistributable Package (x86)
- Microsoft Visual C++ 2015 Redistributable Package (x64)
- Microsoft .NET Framework 4.6 Full
- Microsoft .NET Framework 4.6 Web

These prerequisites install the appropriate technologies on supported target systems.

New Constant for the DialogSetInfo Function

A new constant is available for the nInfoType parameter of the InstallScript function DialogSetInfo:

DLG_INFO_ALTIMAGE_HIDPI—This constant specifies a high DPI image to be displayed in the dialog. High DPI image types supported include BMP, GIF, JPEG, PNG, and TIFF. If transparency is required, image types that support transparency (such as PNG) should be used and szInfoString should specify the name of the image to be displayed (optionally including the path) in the dialog. This parameter applies to all dialogs that display the standard installation image on the left side of the dialog.

When DLG_INFO_ALTIMAGE_HIDPI is passed in nInfoType, the following parameter values are expected:

- szInfoString—The name of an image file to display, optionally including the path. If no path to the file is specific, the file is assumed to be in SUPPORTDIR. If this file does not exist, DialogSetInfo returns ISERR_FILE_NOT_FOUND.
- nParameter—The DPI scaling percentage. For example, pass 200 for a 200% image scale, 150 for a 150% scale, etc. The minimum supported scaling value is 25. If 0 is passed for this value, no image is displayed. If DLG_INFO_ALTIMAGE_REVERT_IMAGE is passed, the previous image used is displayed.

This functionality is available in InstallScript events in the following project types: InstallScript and InstallScript MSI.

For more information, see DialogSetInfo.

Option Added to Disable Architecture Validation

The **None** option has been added to the Architecture Validation options in the Releases explorer to specify the bypassing of build-time architecture validation.

For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.

What's New in InstallShield 2015

New Features

InstallShield includes the following new features.

Support for Windows 10–Based Systems

InstallShield has support for Windows 10.

Targeting Windows 10

On systems with Windows 10, the Windows Installer properties **VersionNT** and **VersionNT64** indicate 603, which was originally introduced as the version number of Windows 8.1. Therefore, it is not possible to create conditions in an .msi package that specifically target Windows 10.

Since Windows Installer 5.0 and Windows 7, DLL custom actions in .msi packages are shimmed to block obtaining the operating system version; the APIs **GetVersion**, **GetVersionEx**, and **RtlGetVersion** return a Windows version of 6.0.6000, which was originally the version number of Windows Vista. Therefore, it is also not possible to obtain the actual version number of Windows from a DLL custom action or from an InstallScript custom action (which is implemented as a DLL).

Because of the aforementioned behavior in Windows Installer, it is not easily possible to detect what version of Windows on which an .msi package is running. In areas where you can specify target system OS requirements, such as the Installation Requirements page in the Project Assistant of Basic MSI and InstallScript MSI projects, the Windows 8.1 option has been renamed as **Windows 8.1 or Windows 10** to reflect the new run-time behavior.

InstallScript, Advanced UI, and Suite/Advanced UI projects provide the ability to properly detect which version of Windows (including Windows 10) is present on a target system. Thus, if your installation needs to target or exclude Windows 10, you may want to use these types of projects to create conditions under which .msi packages should be run based on the actual target system platform.

The InstallShield prerequisites that should be installable on Windows 10 have been updated so that they are installed on those systems if needed. Previously, the prerequisites may not have run by default on those systems.

InstallScript Language Support for Windows 10

The following structure members and predefined constants were added to the InstallScript language:

- **SYSINFO.WINNT.bWin10**—This is a new SYSINFO structure member. If the operating system is Windows 10, this value is TRUE. (This is applicable to InstallScript event-driven code; it is not applicable to custom actions.)
- **ISOSL_WIN10**—This is a new predefined constant that is available for use with the **FeatureFilterOS** function and the SYSINFO structure variable. It indicates that the target system is running Windows 10.

For more information, see FeatureFilterOS function and the SYSINFO structure variable.

Support for Microsoft Visual Studio 2015

InstallShield includes support for Visual Studio 2015 Preview. You can create InstallShield projects from within this version of Visual Studio.

Digital Signing Improvements

InstallShield includes several improvements for digitally signing your installations and files at build time.

Support for SHA-256 Digital Certificates

InstallShield now enables you to use digital certificates that use the SHA-256 hashing algorithm for signing your installations and files at build time.

SHA-256 is favored over SHA-1, which is being deprecated because of the potential for security vulnerabilities. Microsoft announced that Windows will stop trusting items that were signed and timestamped with SHA-1 certificates after January 1, 2016. In addition, certification authorities—the organizations that issue certificates—are phasing out the creation of SHA-1 certificates. Thus, it is recommended that you replace any SHA-1 certificates in your InstallShield projects with SHA-256 certificates. For the latest information and more specific details, check with your certification authority.

In InstallShield, to replace a SHA-1 certificate with a SHA-256 certificate for signing your releases, use the Signing tab in the Releases view to replace the reference to the current certificate with one for a SHA-256 certificate.

If your project is configured to sign with a SHA-256 certificate, InstallShield uses a SHA-256 hash in the signature of the files that it signs at build time. If your project is still configured to sign with a SHA-1 certificate, InstallShield uses a SHA-1 hash; in addition, use of a SHA-1 certificate now triggers build warning -7346 to alert you about the SHA-1 usage.

In earlier versions of InstallShield, InstallShield used a SHA-1 hash in the signature of files when signing with either SHA-1 and SHA-256 certificates.

For details, see Digital Signing and Security.

Ability to Use a Certificate Store for Referencing Certificates

When you are specifying the digital signature information that you want to use for signing your files and installations, InstallShield now lets you reference a certificate store that contains the certificate that you want to use. This support is available as an alternative to specifying a .pfx certificate file on your machine.

To specify whether you want to use a certificate store or a .pfx certificate, use the Digital Certificate File setting on the Signing tab in the Releases view. When you click the ellipsis button (...) in this setting, a new Certificate Selection dialog box opens, enabling you to specify certificate information such as the store name (Personal, Trusted Root Certification Authorities, Enterprise Trust, Intermediate Certification Authorities), the store location (user or machine), and the subject that identifies the specific certificate that you want to use. As an alternative, you can specify on this dialog box the path and file name of a .pfx file that you want to use.

If you configure your project to use a certificate that was imported with password protection into a store, Windows prompts for the password at build time when InstallShield is attempting to sign your project's files. The strong key protection that Windows uses does not permit InstallShield to provide the password to the cryptographic provider.

Certificate store support is also available for signing patches and QuickPatch packages:

- To specify certificate store or .pfx certificate information for a patch, use the Digital Signing tab for a patch configuration in the Patch Design view.
- To specify certificate store or .pfx certificate information for a QuickPatch package, use the Build Settings area of the General Information view in the QuickPatch project. This area includes a Digital Signature tab that includes the new support.

In addition, the command-line tool **iSign.exe**, which is available for signing already-built InstallScript releases, has been updated to include support for using certificates in certificate stores.

To learn more, see:

- Digital Signing and Security
- Certificate Selection Dialog Box
- Signing Tab for a Release (for a release in the Releases view)
- Digital Signature Tab (for a patch configuration in the Patch Design view)
- Digital Signature Tab (in a QuickPatch project)
- iSign.exe

Note that InstallShield no longer has support for signing with .spc and .pvk files. To learn how to convert those files to a .pfx file, see Digital Signing and Security.

Ability to Specify a Program Name for the UAC Dialog Box

The Signing tab in the Releases view includes a new Signature Description setting. Use this setting to specify the text that you want to be displayed to the right of the "Program Name:" label on the UAC dialog box for the Setup.exe files, the .msi file, and other installation files that InstallShield signs at build time. The UAC dialog box opens when an end user launches the signed file and elevated privileges are required.

If you leave the Signature Description setting blank, InstallShield uses the name of the file without its extension as the text on the UAC dialog box.

For more information, see Signing Tab for a Release.

Ability to Digitally Sign Media Header Files with a .pfx File or a Certificate in a Certificate Store

InstallShield now has support for using .pfx files to sign media header files (.hdr files), which are used for the One-Click Install type of installation for InstallScript projects. As an alternative, you can use a certificate in a certificate store to sign media header files.

Previously, it was necessary to sign with .spc and .pvk files instead of .pfx files.

Automation Interface Support for Specifying Digital Certificate Information

The automation interface has support for specifying the .pfx file and the certificate store information. In addition, it also has support for specifying the signature description.

In Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module projects, the ISWiReleases object includes two new read-write string properties.

- **DigitalCertificateInfo**—This property gets or sets either the path to the .pfx file on your machine, or to a string that indicates certificate store details.
- **SignatureDescription**—This property gets or sets the signature description.

To learn more, see ISWiRelease Object.

In Advanced UI and Suite/Advanced UI projects, the ISWiSuiteReleases object includes these same new properties. For more details, see ISWiSuiteRelease Object (Advanced UI and Suite/Advanced UI).

Ability to View Both the 32- and 64-Bit Areas of the Source Machine's Registry on 64-Bit Development Systems

If you are using InstallShield on a 64-bit development system, the Registry view in InstallShield now displays both the 32-bit and 64-bit areas of your machine's registry:

- HKEY_LOCAL_MACHINE\Software
- HKEY_LOCAL_MACHINE\Software\Wow6432Node

This support makes it easier to develop installations on 64-bit machines, since it enables you to drag and drop entries from those source areas to the appropriate areas in the destination pane of this view.

Previously, if you were using InstallShield on a 64-bit development system, the source panes in the Registry view in InstallShield did not show any 64-bit data from the HKLM\Software part of the registry; in addition, the source panes displayed 32-data from the machine's HKLM\Software\Wow6432Node area in the HKLM\Software area.

Note that if you want your installation to install registry data to a 64-bit area of the registry on 64-bit target systems without having it redirected to a 32-bit area, you must place the registry data in a component that is marked as 64 bit. Simply dragging 64-bit data from the source panes in the Registry view to a location in one of the destination panes of the view does not mark the component as 64 bit.

This feature is available in the following project types: Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, MSI Database, MSM Database, and Transform.

For more information, see:

- Developing and Building Installations on 32-Bit vs. 64-Bit Systems
- Dragging and Dropping Registry Entries to Create Registry Keys
- Registry View

Support for Embedding Formatted Expressions in Advanced UI and Suite/Advanced UI Projects for Run-Time Resolution

In various areas of Advanced UI and Suite/Advanced UI projects, you can now embed object expressions to query target systems for information about a file, a registry entry, the operating system, or other details. This enables you to dynamically configure many of the Advanced UI or Suite/Advanced UI settings at run time based on target system–specific conditions. Object expressions use this convention:

```
[@Object(Parameters, ...).Property(Parameters, ...)]
```

Each object expression contains a reference to an object, which is a collection of object-specific properties. Objects and properties may contain parameters.

For example, the following Platform object expression obtains the architecture (x86, x64, IA64, ARM, or Unknown) of the machine on which the Advanced UI or Suite/Advanced UI installation is running:

```
[@Platform.Architecture]
```

The following Registry object expression obtains the value data of the RegisteredOwner value for the HKLM\Software\My Company Name\My Product Name registry key:

```
[@Registry(HKLM\Software\[COMPANY]\[PRODUCT], true).KeyValue(RegisteredOwner)]
```

You can embed object expressions within other formatted expressions, such as within property expressions or other object expressions. In the following expression, a Registry object expression is embedded as part of the parameter to a File object expression.

```
[@File([@Registry(HKLM\Software\MyProduct).KeyValue(MyProductPath)]\MyProduct.exe).Version]
```

If the MyProduct.exe file is in the location that is specified for the MyProductPath value data, the File object expression returns the version of the file. If the file is not found in that location, or if the registry value does not exist, the File object expression returns an empty string.

InstallShield also now enables you to pass literal square brackets as part of strings through the command line to packages in an Advanced UI or Suite/Advanced UI installation. For example, a command line such as **[\[]Text[\]]** resolves as **[Text]** at run time and is passed to the package with the square brackets. Previously, a string enclosed within square brackets was treated as a property that the installation attempted to resolve at run time. The only work around was to use a formatted property expression (such as **[PropertyForSquareBracketString]**) that would be resolved at run time to a property value that included the square brackets.

To learn more, see:

- Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time
- Writing Object Expressions in Advanced UI and Suite/Advanced UI Projects to Search Target Systems
- Object Reference for Expressions in Advanced UI and Suite/Advanced UI Projects
- Feature Object
- File Object
- Parcel Object
- Platform Object
- Registry Object

Ability to Share Common Packages Among Different Advanced UI and Suite/Advanced UI Installations

When you are configuring a package for an Advanced UI or Suite/Advanced UI installation, you can now mark it as shared. The shared package functionality helps to ensure that if two or more Advanced UI or Suite/Advanced UI installations are sharing a package, the package remains on the target system until all of the Advanced UI and Suite/Advanced UI products are removed.

To mark a package that is selected in the Packages view of an Advanced UI or Suite/Advanced UI project as shared, select Yes for the new Shared setting. In addition, ensure that the package GUID is the same across all Advanced UI and Suite/Advanced UI projects that are sharing the package; the Package GUID setting in the Packages view is where you can view and modify the GUID of a package.

Before this built-in shared support was available, unexpected results may have occurred in various scenarios. For example, in some scenarios, if two different Advanced UI or Suite/Advanced UI installations installed a shared package on a target system and then only one was uninstalled, the shared package was removed. The remaining Advanced UI or Suite/Advanced UI product may have been unusable because of the missing shared package. In other scenarios, the shared package was erroneously left behind on a target system, even though no Advanced UI or Suite/Advanced UI product remained.

This feature is available for the following types of packages in Advanced UI and Suite/Advanced UI projects: .msi, .msp, .exe, .appx, InstallScript, Basic MSI project, and InstallScript project.

For more details, see:

- Sharing Common Packages Among Different Advanced UI and Suite/Advanced UI Installations
- Packages View

Ability to Override the Product Name, Product Version, and Suite GUID for a Release in an Advanced UI or Suite/Advanced UI Project

InstallShield now lets you override the product name, product version, and Suite GUID values that are specified in the General Information view of your Advanced UI or Suite/Advanced UI project with a new value for a selected release. To override the General Information view values, use the new settings that are available on the Build tab for a release in the Releases view: Product Name, Product Version, and Suite GUID.

For details, see Build Tab for a Release.

Ability to Override the Values of Path Variables for a Release in an Advanced UI or Suite/Advanced UI Project

InstallShield now lets you override the values of your project's custom path variables (standard path variables, environment path variables, and registry path variables) for each release in your project. This functionality enables you to essentially replace certain files and folders in your project with others at build time, depending on the particular release that you are building.

For example, you might want to use this functionality to swap out UI elements such as images and EULAs for different releases in your project; this would enable you to easily generate installations for different editions or branded versions of your product.

To override one or more path variables in your project, use the Path Variables Overrides setting, which is on the Build tab for a release in the Releases view.

To learn more, see the following:

- Overriding the Value of a Custom Path Variable for a Release

- Path Variable Overrides setting

Automation Interface Support for Advanced UI and Suite/Advanced UI Projects

InstallShield now enables you to automate most development and build processes of Advanced UI and Suite/Advanced UI projects (.issuite files) through the automation interface without having to directly open the InstallShield interface and make changes in different views. The automation interface provides programmatic access to various areas of Advanced UI and Suite/Advanced UI projects; it exposes a COM interface that you can call from many languages to create, edit, and build Advanced UI or Suite/Advanced UI projects.

The top-level automation object for Advanced UI and Suite/Advanced UI projects is the ISWiProject object. Sample VBScript code that opens, modifies, saves, and closes a project begins and ends like this:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("ISWiAutoSuite22.ISWiProject")
pProj.OpenProject "C:\InstallShield 2015 Projects\Project1.issuite"
' perform changes here
pProj.SaveProject
pProj.CloseProject
```

By calling methods, getting and setting properties, and accessing collections, you can also add features and packages to your project, configure conditions, and more. For more information, see the following sections of the documentation:

- Automation Objects for Advanced UI and Suite/Advanced UI Projects
- Automation Collections for Advanced UI and Suite/Advanced UI Projects

Note that the ProgID for the Advanced UI and Suite/Advanced UI automation interface is ISWiAutoSuiteAutomation Interface Version.ISWiProject; the ProgID for other project types is simply ISWiAutoAutomation Interface Version.ISWiProject.

New InstallShield Prerequisites for Microsoft Visual C++ 2015, .NET Framework 4.6, and More

InstallShield includes new InstallShield prerequisites that you can add to your projects:

- Microsoft Visual C++ 2015 Redistributable Package (x64)
- Microsoft Visual C++ 2015 Redistributable Package (x86)
- Microsoft Visual C++ 2013 Redistributable Package (x86)
- Microsoft Visual C++ 2013 Redistributable Package (x64)
- Microsoft .NET Framework 4.6 Full
- Microsoft .NET Framework 4.6 Web
- Microsoft .NET Framework 4.5.2 Full
- Microsoft .NET Framework 4.5.2 Web
- Microsoft SQL Server 2012 Express SP2 (x86)
- Microsoft SQL Server 2012 Express SP2 (x86 & x64Wow)
- Microsoft SQL Server 2012 Express SP2 (x64)
- Microsoft SQL Server 2012 Express SP2 LocalDB (x86)

- Microsoft SQL Server 2012 Express SP2 LocalDB (x64)
- Microsoft SQL Server 2012 Express SP2 Management Objects (x86)
- Microsoft SQL Server 2012 Express SP2 Management Objects (x64)
- Microsoft SQL Server 2012 Express SP2 System CLR Types (x86)
- Microsoft SQL Server 2012 Express SP2 System CLR Types (x64)
- Internet Explorer 11.0 for Windows 7 (x86)
- Internet Explorer 11.0 for Windows 7 and Windows Server 2008 R2 (x64)
- Microsoft ReportViewer 2012

These prerequisites install the appropriate technologies on supported target systems.

The Microsoft SQL Server 2012 Express SP2 prerequisites replace the Microsoft SQL Server 2012 Express SP1 prerequisites.

New Predefined System Searches for Internet Explorer 10 and 11

InstallShield has new predefined system searches that check target systems for Internet Explorer 10 or Internet Explorer 11. If your installation or product requires either of those versions, you can use the System Search view or the Installation Requirements page in the Project Assistant to add one of these system searches to your project. When end users launch your installation, Windows Installer checks the target system to see if the requirements are met; if they are not met, the installation displays the error message that is defined for the system search.

This change is available in Basic MSI and InstallScript MSI projects.

Support for Creating Virtual Applications for Microsoft App-V 5.0 SP3; Additional App-V Package Improvements

InstallShield and the Microsoft App-V Assistant in InstallShield include support for creating virtual applications that can run on Microsoft App-V 5.0 SP3 clients. In addition, the Microsoft App-V Assistant includes new settings and capabilities—some of which apply to App-V 5.0 SP3, and some of which apply to earlier versions of App-V.

New InstallShield Prerequisites for App-V 5.0 SP3

InstallShield includes new InstallShield prerequisites that you can add to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects:

- Microsoft App-V 5.0 SP3 Desktop Client (x86)
- Microsoft App-V 5.0 SP3 Desktop Client (x64)

The redistributable files for these InstallShield prerequisites are not available for download from within InstallShield, since you must obtain them from Microsoft. Once you obtain one of the redistributables from Microsoft, place it in the location that is displayed when you are editing the prerequisite in the InstallShield Prerequisite Editor.

Ability to Map Files into the Virtual File System Instead of a Primary Application Directory

The Microsoft App-V Assistant now enables you to configure your App-V package to map files into the virtual file system (VFS). This support is available for App-V 4.x and 5.x packages.

To specify whether you want to map the files to the VFS or use a primary application directory, use the Files page in the Microsoft App-V Assistant. The More Options area on this page has a new File Mapping link. When you click this new link, a new File Mapping dialog box opens, enabling you to select the appropriate option.

The File Mapping link and dialog box replace the Primary Application Directory link and dialog box, which previously enabled you to specify primary application directory.

To learn more, see File Mapping Dialog Box.

Ability to Create App-V 5.x Packages that Have Full Write Permissions to the Virtual File System

The Microsoft App-V Assistant now enables you to specify whether you want an App-V 5.x package that you are creating to have full write permissions to the virtual file system (VFS). To specify whether you want to use this support, use the new Allow full write permission to the VFS check box. This check box is on the File Mapping dialog box, which is available from the Files page in the Microsoft App-V Assistant when you click the File Mapping link in the More Options area.

To learn more, see File Mapping Dialog Box.

Ability to Configure Advanced COM Isolation Settings for App-V 5.x

The Microsoft App-V Assistant now enables you to configure advanced settings for COM isolation. This support is available for App-V 5.x packages.

To configure the new settings, use the Package Information page in the Microsoft App-V Assistant. The More Options area on this page has a new Isolation Settings link. When you click this new link, a new Isolation Settings dialog box opens. This dialog box enables you to specify whether you want to isolate the COM objects from the local system, or allow them to interact with the local system. This dialog box also lets you indicate whether you want to isolate named objects from the local system, or allow them to interact with the local system.

To learn more, see Isolation Options Dialog Box (for a Package).

The Microsoft App-V Assistant is available in the Premier Edition of InstallShield.

Enhancements

Performance Improvement for the Files and Folders View

InstallShield has been enhanced to more quickly load the Files and Folders view of large projects.

Ability to Filter Items in the Files and Folders View by Feature

The Files and Folders view now contains a View Filter that lists each of the features in your project, as well as an All Application Data option. You can use this filter to show and hide files and folders in the destination panes of this view:

- To see in this view only files and folders that belong to a particular feature, select that feature in the View Filter list.
- To add a file or folder to a specific feature, select that feature in the View Filter list. Then add the file or folder to the appropriate location in the **Destination computer's folders** pane.
- To see all of the files and folders that are in your project, select the All Application Data option in the View Filter list.

The new View Filter replaces the previous **Add new components to the feature** filter, which did not have support for showing and hiding files and folders in the view according to the selected feature.

To learn more, see Files and Folders View.

Ability to Access the Installer Session from PowerShell Scripts in Basic MSI and InstallScript MSI Installations

The PowerShell custom action support has been enhanced. It now includes support for several cmdlets that let you interact with the running Basic MSI or InstallScript MSI installation at run time. The cmdlets enable you to get and set Windows Installer properties, expand the value of formatted expressions, and write information to the log file.

With this revised implementation, you can use the Windows Installer property **IS_CLR_VERSION** to identify a semicolon-delimited list of .NET Framework versions that the custom action should attempt to load to run your PowerShell script.

To learn more, see Calling a PowerShell Custom Action.

Ability to Configure Major Upgrades in Transform Projects and MSI Databases

Transform projects and MSI Database projects now include the Upgrades view, which you can use to create major upgrades. To add an upgrade entry in a Transform project or in direct-edit mode for an MSI Database project, open the Upgrades view. Right-click the Upgrade Windows Installer Setup node and then click Add Major Upgrade Item. Then configure its settings in the right as needed.

To learn more, see:

- Upgrades View
- Adding a Major Upgrade Item
- Major Upgrades
- Upgrade Considerations
- Preventing the Current Installation from Overwriting a Future Major Version of the Same Product

Improvements to the ISHiddenProperties Property for Preventing Passwords from Being Written in Advanced UI and Suite/Advanced UI Log Files

The **ISHiddenProperties** property stores a semicolon-delimited list of case-sensitive property names whose values you do not want to be written to the debug log files. This property enables you to prevent properties that contain passwords and other sensitive information from being logged. The **ISHiddenProperties** property has been improved. You can now use this property for prevention of logging for values in the following scenarios:

- The end user sets the value of the property through command line when launching the Advanced UI or Suite/Advanced UI **Setup.exe** file.
- The property is configured through a command line that the Advanced UI or Suite/Advanced UI installation passes to a package. This is configurable in the Packages view, on the Common tab, in the Operation area.

Previously, **ISHiddenProperties** prevented logging only values that would be logged by changing the property value.

For more information, see Preventing a Property Value from Being Written in Advanced UI and Suite/Advanced UI Debug Log Files.

Improvements to the Interpretation of Ampersands in the Wizard Interface of Advanced UI and Suite/Advanced UI Installations

The interpretation of ampersands in certain areas of the wizard interface of Advanced UI and Suite/Advanced UI projects has been modified.

If you use an ampersand (&) in any of the following areas of the wizard interface of your installation, the installation now displays the ampersand as a literal character instead of interpreting it as a prefix character for a keyboard shortcut. The change also applies if any of these strings include a property that resolves to a value that contains an ampersand.

- The string in the header area of a wizard page or secondary window—This is configured in the Title setting for a wizard page or window in the Wizard Interface view.
- The caption bar of wizard pages—This is configured in the Wizard Caption setting for the Wizard Pages node in the Wizard Interface view.
- The supplemental explanation area of a command link control—This is configured in the Note setting for a wizard page or window.
- The alternate text for an image control—This is configured in the Alt Text setting of an image control on a wizard page or window.

For most label controls, True is now selected by default for the SS_NOPREFIX subsetting under the Style setting; previously False was selected by default. A True value for this subsetting ensures that any ampersands that are included in the string entries for such controls are not inadvertently interpreted as keyboard shortcuts, and that ampersands in the control's strings are displayed as ampersands in the wizard interface. This applies to the built-in default wizard pages and windows, as well as the predefined wizard pages. The only exceptions to this SS_NOPREFIX subsetting change are label controls that may contain keyboard shortcuts. For example, the default predefined Web Deploy wizard page has several text boxes with corresponding labels that may contain keyboard shortcuts. Therefore, False continues to be default value for the SS_NOPREFIX subsetting of such controls.

The InstallShield Help Library has a new Designating Keyboard Shortcuts and Using Ampersands (&) in the Wizard Interface help topic that explains how to configure your project to support the use of ampersands and keyboard shortcuts in your wizard interface.

Note that when you upgrade an InstallShield 2014 or earlier Advanced UI or Suite/Advanced UI project to InstallShield 2016, many of the aforementioned changes for ampersand interpretation are made automatically made; however, some changes are not made. To learn more, see Upgrading Projects from InstallShield 2014 or Earlier.

Enhanced File-Related Types of Condition Checks Can Search Multiple Target System Paths in Advanced UI and Suite/Advanced UI Installations

When you are building a conditional statement for an exit, detection, eligibility, feature, or wizard interface condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can use the following file-related types of condition checks:

- **File Exists**—This type checks target systems for the presence of a particular file.
- **File Comparison**—This type checks target systems for specific information—date, version number, or component—for a particular file.

Both of these condition checks have been expanded to enable you to define conditions that check for the file in multiple paths on target systems. Previously, each condition could check only one specific location on target systems.

To enable this support, the existing Path setting for these condition checks has been expanded to let you enter only the name of the file (that is, optionally now without the path); if you leave out the path, you can use the new Search Path setting to specify a semicolon-delimited list of paths that you want the installation to check at run time for the specified file. You can include in this setting one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions.

For example, to search for the specified file in all directories that are defined for the PATH environment variable on a target system, as well as the path that is defined in the registry, enter the following value in the Search Path setting:

[%PATH];[@Registry(HKLM\SOFTWARE\MyPath).KeyValue(MyValue)]

For more details, see:

- File Exists Condition Settings
- File Comparison Condition Settings

Improvements for Importing InstallShield Prerequisites with Certain Types of File-Related Condition Checks in Advanced UI and Suite/Advanced UI Projects

Advanced UI and Suite/Advanced UI projects now have support for the following conditions that InstallShield prerequisites support:

- If you use the condition **A file does or does not exist** or **A file with a certain date exists** in a prerequisite and if you reference a registry key within square brackets for part of the path to the file, a Basic MSI, InstallScript, or InstallScript MSI installation resolves the registry key with the registry value. Now that Advanced UI and Suite/Advanced UI installations have expanded formatted-expression support for conditions, these types of prerequisite conditions are properly imported into Advanced UI and Suite/Advanced UI projects and resolved at run time. Previously, the registry key within square brackets was not resolved at run time; therefore, these types of conditions always evaluated as false.
- If you use the condition **A file does or does not exist** or **A file with a certain date exists** in a prerequisite and if you omit the path to the file, a Basic MSI, InstallScript, or InstallScript MSI installation searches for the specified file in all directories that are defined for the PATH environment variable on the target system. Now that Advanced UI and Suite/Advanced UI installations have expanded formatted-expression support for conditions, these types of prerequisite conditions are properly imported into Advanced UI and Suite/Advanced UI projects and resolved at run time. Previously, the path to the file could not be resolved; therefore, this type of condition always evaluated as false.

The InstallShield prerequisites for Oracle Instant Client are examples of prerequisites that use the file-related conditions. Now if you import these types of InstallShield prerequisites into Advanced UI or Suite/Advanced UI projects, those file-related conditions are configured properly in the Detection Condition setting of the Packages view.

For more information, see Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project.

Ability to Upgrade a Project Through the Automation Interface

The automation interface includes support for upgrading a project from an earlier version of InstallShield to the current version. To upgrade a project through the automation interface, use the ForceUpgrade method of the ISWiProject object.

This enhancement is available for the following project types: Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, and Suite/Advanced UI.

For more information, see:

- ISWiProject Object
- ISWiProject Object (Advanced UI and Suite/Advanced UI)

Ability to Select the Virtual Machine Configuration for a Release Through the Automation Interface

The automation interface has support for selecting the virtual machine configuration for a given release; this enables you to specify which configuration you want to use for distributing the selected release to a virtual machine (VM) at build time.

In Basic MSI, InstallScript, and InstallScript MSI projects, the ISWiReleases object includes several new read-write properties. In Suite/Advanced UI projects, the ISWiSuiteRelease object includes the same new read-write properties.

- **VMConfig**—This property gets or sets the name of the group of VM configuration settings that you want to use when distributing a release to a VM.
- **VMSnapShot**—This property optionally gets or sets the name of the snapshot that you want to use for distributing the release.

  If no snapshot is specified for the VM configuration, InstallShield does not revert to a specific snapshot. InstallShield powers on the VM without reverting it to a specific snapshot, and copies your installation to the VM.
- **VMMachineUserName**—This property gets or sets the user name for the VM configuration.
- **VMMachinePassword**—This property gets or sets the password for the VM configuration.
- **VMStageMachineCopyPath**—This property gets or sets the location on the VM where the release is to be distributed. The last subfolder in the path can be a path that InstallShield creates on the VM; the other folders in the path must already exist.
- **VMStagePostBuild**—This Boolean property indicates whether you want InstallShield to automatically distribute the selected release each time that it is successfully built.

This enhancement is available in the Premier edition of InstallShield.

To learn more, see:

- ISWiRelease Object
- ISWiSuiteRelease Object (Advanced UI and Suite/Advanced UI)

Enhancement to ISICE11

InstallShield internal consistency evaluator 11 (ISICE11) has been improved. If an executable file in your project includes a valid manifest that includes an asm.v2 entry for the trustInfo element, ISICE11 no longer occurs during validation. Previously, the asm.v3 entry was checked, but not the asm.v2 entry.

To learn more, see ISICE11.

Enhanced Dialogs View in Basic MSI, DIM, and Merge Module Projects: All Behavior Settings for Each Control Are Shown in a Single Grid

The Dialogs view has been enhanced. All of the settings that are available for configuring the behavior (events, subscriptions, and conditions) of each user interface control on a run-time dialog are now displayed in a single grid. Previously, the settings were available through separate Events, Subscriptions, and Conditions tabs that were displayed in the bottom-right corner of the view.

To add an event, a subscription, or a condition to a control, in the Dialogs view, select the Behavior node under the dialog that contains the control. The controls that are used on the dialog are displayed in the center pane; select the control that you want to configure. Then use the Events, Subscriptions, and Conditions settings that are displayed in the right pane to configure the appropriate behavior.

This enhancement applies to the following project types: Basic MSI, DIM, Merge Module.

To learn more, see:

- Editing Dialog Behavior in Basic MSI Projects
- Using Control Events in Basic MSI Dialogs
- Using Subscriptions in Basic MSI Dialogs

Enhanced Folder Views Show Total Count of Each Item in the Current Project

The folder views in InstallShield now enable you to see a quick summary of the contents of your project.

For example, the Application Data view—which is the folder view that contains the Files and Folders view and the Redistributables view in some types of projects—indicates the number of files, the number of merge modules, and the number of InstallShield prerequisites that are in the current project. The System Configuration view—which is the folder view that contains views such as Shortcuts and Registry— shows summary data such as the number of shortcuts, the number of registry keys, and the number of registry values.

New Machine-Wide Setting for Removing Unused Components Automatically

The Files View tab on the Options dialog box in InstallShield has a new **Clean up unused components** check box. This check box lets you specify whether you want InstallShield to remove unused components from your project automatically.

When this check box is selected, you delete all of the files in a component, and that component is not required elsewhere, that component is deleted automatically.

This new check box is a machine-wide setting. This check box is cleared by default; therefore, unused components are not removed automatically by default.

For more information, see Files View Tab.

Enhanced Behavior for Deleting Custom Actions that Use a File in the Binary Table

When you are deleting from your project a custom action that uses a file in the **Binary** table, InstallShield now determines whether the file in the **Binary** table is referenced by any other custom actions in the project. If it is not referenced by any other custom actions, InstallShield prompts you to specify whether you want to delete the entry from the **Binary** table while also deleting the custom action:

- The No button, which is the default selection, enables you to delete only the custom action.
- The Yes button enables you to delete both the custom action and the **Binary** table entry.

- The Cancel button enables you to avoid deleting the custom action and the **Binary** table entry.

If one or more other custom actions in the project reference the file in the **Binary** table, InstallShield deletes only the custom action that you chose to delete; it does not delete the **Binary** table entry. In this scenario, InstallShield does not display any prompt when you are deleting the custom action.

Previously, when you were deleting a custom action in the aforementioned scenario, InstallShield always displayed a prompt asking whether you wanted to delete the entry in the **Binary** table—regardless of whether any other custom actions in the project referenced it. The default button on this prompt was Yes, which resulted in both the custom action and the **Binary** table entry being deleted; this default behavior led to some users inadvertently deleting the **Binary** table entry.

This enhancement is applicable to the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, and Transform.

New Standalone Build Registry Entries Indicate InstallShield Version Information

The Standalone Build installation now installs the same InstallShield version–related registry values that the InstallShield installation installs under HKEY_LOCAL_MACHINE\SOFTWARE\InstallShield*VersionNumber*\Professional. This makes it possible to quickly verify whether all of the development and build machines in your environment are using the same versions of the product. The pertinent registry values are:

- **Product Code**—This registry value differs, depending on whether the InstallShield IDE or the Standalone Build is installed. It is updated for major upgrades of these tools.
- **MPIndicator**—If applicable, this registry value indicates the maintenance pack—for example, *Service Pack 1*.
- **SP**—If applicable, this registry value indicates the number of the service pack—for example, *1*.

For InstallShield 2016 and the InstallShield 2016 Standalone Build, the registry values are installed under the following key:

HKEY_LOCAL_MACHINE\SOFTWARE\InstallShield\Automation Interface Version\Professional

What's New in InstallShield 2014 SP1

InstallShield includes the following improvements.

Updates to Portuguese Run-Time Strings

The default run-time strings for Portuguese (Brazil) and Portuguese (Portugal) have been updated to reflect the Portuguese Language Orthographic Agreement of 1990 that finishes going into effect at the end of 2014, per a 6-year transition period starting in 2009.

What's New in InstallShield 2014

New Features

InstallShield includes the following new features.

New InstallShield Prerequisites for .NET Framework 4.5.1

InstallShield includes new InstallShield prerequisites that you can add to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects:

- Microsoft .NET Framework 4.5.1 Full
- Microsoft .NET Framework 4.5.1 Web

These prerequisites install the .NET Framework 4.5.1 on supported target systems.

Microsoft SQL Server 2014 Support

InstallShield now includes support for running SQL scripts on SQL Server 2014 database servers. In addition, InstallShield includes SQL Server 2014 in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports.

If your installation targets SQL Server 2014, the SQLBrowse run-time dialog that is displayed when end users choose to browse for a database server can now list instances of SQL Server 2014, SQL Server 2014 Express, and SQL Server 2014 Express LocalDB. In addition, the SQLBrowse run-time dialog that is displayed when end users choose to browse for a database catalog can now list catalogs on the specified SQL Server 2014 database server.

This support is available in the following project types: Basic MSI, DIM, InstallScript, and InstallScript MSI.

Microsoft SQL Server 2014 Prerequisites

InstallShield includes several new SQL Server 2014–related InstallShield prerequisites that you can add to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects:

- Microsoft SQL Server 2014 Express RTM (x64)
- Microsoft SQL Server 2014 Express RTM (x86 & x64Wow)
- Microsoft SQL Server 2014 Express RTM (x86)
- Microsoft SQL Server 2014 Express RTM LocalDB (x64)
- Microsoft SQL Server 2014 Express RTM LocalDB (x86)

These InstallShield prerequisites install the technology on supported target systems.

New InstallShield Prerequisites for App-V 5.0 SP2

InstallShield includes new InstallShield prerequisites that you can add to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects:

- Microsoft App-V 5.0 SP2 Desktop Client (x86)
- Microsoft App-V 5.0 SP2 Desktop Client (x64)

The redistributable files for these InstallShield prerequisites are not available for download from within InstallShield, since you must obtain them from Microsoft. Once you obtain one of the redistributables from Microsoft, place it in the location that is displayed when you are editing the prerequisite in the InstallShield Prerequisite Editor.

New InstallShield Prerequisites for Microsoft Visual C++ 2012 Update 4

InstallShield includes new InstallShield prerequisites that you can add to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects:

- Microsoft Visual C++ 2012 Update 4 Redistributable Package (x86)
- Microsoft Visual C++ 2012 Update 4 Redistributable Package (x64)

These prerequisites install the various technologies on supported target systems.

Support for Checking for Updates from Maintenance Mode of Advanced UI and Suite/Advanced UI Installations

Advanced UI and Suite/Advanced UI installations now have support for enabling end users to check for updates to your product by clicking a new Update button on the maintenance wizard page. If an update is available from your Web site and the end user chooses to obtain it, the installation downloads it, verifies its digital signature, and launches it.

To use this functionality, your Advanced UI or Suite/Advanced UI project must have support for automatic updates. At build time, InstallShield creates a metadata file called **isupdate.xml** file and places it in an UpdateMetadata subfolder in your project's build folder. This metadata file identifies the different versions of your Advanced UI or Suite/Advanced UI installation and their download locations. When you are ready to release the update for your product, you upload to your site this **isupdate.xml** file as well as your updated Advanced UI or Suite/Advanced UI installation.

At run time, if the end user is running maintenance mode, the Suite engine checks the metadata file for updates. If an update is available for download, the Suite engine sets the new **ISUpdateAvailable** property to 1 and displays a new MaintenanceUpdateWelcome wizard page, a maintenance wizard page that includes an Update button. In addition, the Suite engine sets the new **ISUpdateVersion** property equal to the version of the Suite/Advanced UI or Advanced UI installation that is available for download.

If an update is not available, the Suite engine displays the standard MaintenanceWelcome wizard page that does not have an Update button.

For more details, see:

- Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation
- Advanced UI and Suite/Advanced UI Property Reference

Ability to Include Support Files in Subfolders

InstallShield now lets you specify custom folder structures for support files. To add a subfolder under one of the language-specific nodes or to the language-independent node in the Support Files view, right-click the node and then click New Folder. InstallShield adds a subfolder and enables you to rename it as needed. You can also add a nested folder structure. To add files to a subfolder, select it and right-click in the right pane, and then click Insert Files. At run-time, the installation copies the support folders and files to **SUPPORTDIR**, a temporary directory on target systems, to make them available during your product's installation process. The support folders and files are deleted when the installation is complete.

The improved Support Files view is available in the following project types: Advanced UI, Basic MSI, Suite/Advanced UI, InstallScript, and InstallScript MSI.

To learn more, see Adding Support Files.

The automation interface also includes subfolder functionality for support files. The ISWiSetupFile object includes a new read-write Target property that you can use to specify or obtain the path of the support file's target directory within **SUPPORTDIR**. For more information, see ISWiSetupFile Object.

Support for Deploying Web Deploy Packages to IIS Web Servers and the Cloud

Suite/Advanced UI installations now have built-in support for deploying Web Deploy packages to IIS Web servers and the cloud. The Web Deploy packages can be created through IIS or a Web application development environment such as Visual Studio.

To add a Web Deploy package to your Suite/Advanced UI project, use the Packages view. When you add a package and select it in the Packages view, InstallShield displays package-related settings in the right pane. The Configuration area on the Common tab of the right pane includes several configuration-related settings for Web Deploy, such as the destination—which is often a remote server—and credentials. In addition, the Configuration area also includes settings for the parameters that are defined in the Web Deploy package's parameters XML file. By default, InstallShield uses Suite/Advanced UI properties for all of the configuration settings so that end users can set them at run time through the wizard interface or the command line.

When you add a Web Deploy package to your project, InstallShield adds a predefined Web Deploy wizard page to your project for that package. The Web Deploy wizard page enables the end user to specify whether the package will be deployed to a local IIS server, a remote server, or a cloud-based server. It also enables the end user to load the configuration settings from a publisher profile file (.publishsettings). If you want end users to be able to configure the parameters that are defined in your Web Deploy package's parameters XML file, you can add controls to the wizard interface as needed, and associate those controls with the properties that are specified for the corresponding parameter settings in the Packages view.

Note that Web Deploy packages are typically intended to behave similarly to first-time installations or to install over existing earlier or identical versions. As a side effect, Web Deploy packages do not handle uninstallation. Thus, Suite/Advanced UI installations do not offer maintenance for this type of package. It is recommended that you avoid creating a Suite/Advanced UI installation that combines one or more Web Deploy packages with one or more traditional packages (that do offer maintenance) as primary packages.

This feature is available in the Premier edition of InstallShield.

For more information, see:

- Adding a Web Deploy Package to a Suite/Advanced UI Project
- Packages View

Support for Including InstallShield Projects as Packages in Suite/Advanced UI Projects

InstallShield now lets you add InstallShield projects (.ism) as packages in Suite/Advanced UI projects. The InstallShield projects can be either Basic MSI projects or InstallScript projects; the projects must have been saved in the same version of InstallShield as the Suite/Advanced UI project.

When you start a build of a release for a Suite/Advanced UI project that includes one or more InstallShield project packages, InstallShield first builds the designated releases in the associated InstallShield projects, and includes them as packages in the Suite/Advanced UI installation that it generates. InstallShield adds the output of the Basic MSI and InstallScript projects to the Suite/Advanced UI installation as .msi and .hdr packages.

Use the Packages view to add an InstallShield project as a package in your Suite/Advanced UI project. This view is also where you identify which product configuration (for a Basic MSI project package) and which release (for both Basic MSI project packages and InstallScript project packages) that you want to include as a package in the Suite/Advanced UI project.

This feature is available in the Premier edition of InstallShield.

For more information, see:

- Adding an InstallShield Project (.ism) as a Package in a Suite/Advanced UI Project
- Packages View

Support for InstallScript Actions in Suite/Advanced UI Installations

Suite/Advanced UI installations now have built-in support for launching InstallScript actions to perform various run-time tasks that are outside the scope of the packages that you are including in the installation. These Suite/Advanced UI–based InstallScript actions function much like InstallScript custom actions in Basic MSI projects.

You can use the InstallScript view that is available now in Suite/Advanced UI projects to add InstallScript files (.rul) to your project and write functions using the InstallScript language. To have your Suite/Advanced UI action call an InstallScript function, you must prototype the function with the **export** keyword, just as you would for InstallScript custom actions in Basic MSI projects.

Once you have added InstallScript code to your project, use the Events view to add an InstallScript action that calls your code. Next, you can schedule when at run time you want the action to be launched. For example, you can use the Events view to schedule an InstallScript action to run during one or more of the built-in events that the Suite/Advanced UI engine manages.

Note that there are some limitations to the use of InstallScript in Suite/Advanced UI projects. For example, InstallScript UI–related functions and InstallScript run-time path variables (such as PROGRAMFILES and WINDIR) are not available for use in a Suite/Advanced UI project.

This feature is available in the Premier edition of InstallShield.

To learn more, see:

- Using Actions to Extend the Behavior of a Suite/Advanced UI Installation
- Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation
- Configuring a Suite/Advanced UI Action's Settings
- Scheduling a Suite/Advanced UI Action
- Suite/Advanced UI and Advanced UI Interaction Functions

Support for Managed-Code Actions in Suite/Advanced UI Installations

Suite/Advanced UI installations now have built-in support for launching managed-code actions to perform various run-time tasks that are outside the scope of the packages that you are including in the installation. The new action type calls a public method in a .NET assembly that is written in managed code such as Visual Basic .NET or C#. These Suite/Advanced UI–based InstallScript actions are functionally equivalent to DLL actions in Suite/Advanced UI installations. That is, Suite engine looks for one parameter on the managed function that it calls. This parameter is an ISuiteExtension interface passed as a System.Object. The return type should be of some 32-bit integer type.

To add a managed-code action to a Suite/Advanced UI project, use the Events view.

This feature is available in the Premier edition of InstallShield.

For more information, see:

- Using Actions to Extend the Behavior of a Suite/Advanced UI Installation
- Working with a Managed-Code Action in a Suite/Advanced UI Installation
- Configuring a Suite/Advanced UI Action's Settings

- Scheduling a Suite/Advanced UI Action

Ability to Launch Suite/Advanced UI Event Actions that End Users Trigger When Using Wizard Interface Controls

Suite/Advanced UI installations include support for run-time actions that are defined in the Events view and that are launched when an end user clicks or performs some other action on a wizard interface control. For example, this capability enables your installation to run executable files, call DLL functions, run InstallScript code, call managed methods in a managed assembly, run PowerShell scripts, or set a Suite/Advanced UI property when an end user clicks a button in the wizard interface.

To add an action to your project, use the Events view.

To schedule an action to run when a wizard interface control is clicked or used, select the control that you want to configure in the Wizard Interface view, and then use the Click setting or an action-related setting under the Events setting to select the action that you want to launch.

This feature is available in the Premier edition of InstallShield.

For more details, see:

- Configuring an Action for an Element in the Wizard Interface
- Using Actions to Extend the Behavior of a Suite/Advanced UI Installation

Ability to Launch Suite/Advanced UI Actions When Wizard Pages and Windows Are Shown or Hidden

Suite/Advanced UI installations include support for run-time actions that are launched when a wizard page or secondary window is opened or closed. This capability enables your installation to perform any initialization such as dynamically retrieving values for a combo box. It also lets you schedule actions to occur before the wizard interface is displayed at run time.

To add an action to your project, use the Events view.

To schedule an action to run when a wizard page opens or closes, select the wizard page that you want to configure in the Wizard Interface view, and then select the appropriate action in the Page Entered setting or the Page Exited setting.

To schedule an action to run when a secondary window opens or closes, select the secondary window that you want to configure in the Wizard Interface view, and then select the appropriate action in the Window Entered setting or the Window Exited setting.

This feature is available in the Premier edition of InstallShield.

For more details, see:

- Configuring an Action for an Element in the Wizard Interface
- Using Actions to Extend the Behavior of a Suite/Advanced UI Installation

New Property for Preventing Passwords from Being Written in Advanced UI and Suite/Advanced UI Installation Log Files

If you launch an Advanced UI or Suite/Advanced UI **Setup.exe** file with the /debuglog parameter, the Suite engine generates a debuglog file. By default, the debug log file includes the values of changed Advanced UI or Suite/Advanced UI properties.

In some cases, you may want to prevent the Suite engine from writing the values of specific properties to the debug log file. For example, you may want to prevent properties that contain passwords and other sensitive information from being logged. To enable this scenario, Advanced UI and Suite/Advanced UI installations support a new property called **ISHiddenProperties**. You can set this property to a semicolon-delimited list of case-sensitive property names whose values you do not want to be written to debug log files.

Note that **ISHiddenProperties** is useful for prevention of logging only for values that would be logged by changing the property value. If the property is logged any other way (such as through ISuiteExtension::LogInfo), the Suite engine cannot prevent the logging. Therefore, any code that you create to write a property value to the log file should read **ISHiddenProperties** to see whether the property value should be logged.

For more information, see Preventing a Property Value from Being Written in Advanced UI and Suite/Advanced UI Debug Log Files.

Enhanced Substitution Syntax for Advanced UI and Suite/Advanced UI Projects

Various areas of Advanced UI and Suite/Advanced UI projects include support for enclosing property names in square brackets to indicate that the properties should be replaced with appropriate values at run time. The run-time syntax has been enhanced to support several additional types of substitution:

- **Special characters**—Preceding a special character with a backslash and enclosing the resulting string in square brackets resolves the special character at run time. For example, an entry such as **[\[]Text[\]]** resolves as **[Text]**.
- **Null character**—A tilde that is enclosed in square brackets—that is, **[~]**—resolves as a null character embedded in the string.
- **Environment variables**—Preceding the name of an environment variable with a percent sign and enclosing the resulting string in square brackets resolves the environment variable at run time. For example, **[%PATH]** resolves as the value of the PATH environment variable.
- **Recursive property resolution**—Surrounding a property resolution with additional square brackets resolves the resolved value of a property. For example **[[PROPERTY1]]** resolves to the value of the property whose name is stored in **PROPERTY1**; if **PROPERTY1** holds **PROPERTY2**, the resolved value is the value stored in **PROPERTY2**.

For more information, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

Support for Removing Files and Folders

InstallShield now has built-in support that makes it easy to specify files and folders that you want to be removed from target systems at run time. This file and folder removal capability is useful for scenarios such as removing application-created files that your installation does not otherwise track.

You can schedule the removal of a file or folder for one of the following events:

- When the file or folder's component is being installed
- When the file or folder's component is being uninstalled
- When the file or folder's component is being installed or uninstalled

Note that if the item to be removed is a folder, that folder is removed only if it is empty.

InstallShield offers different methods for configuring a file or folder removal in a project:

- In the Files and Folders view, select the destination folder that contains the file or folder that you want to be removed. Then right-click in the **Destination computer's files** pane and click Add File Removal.
- In the Setup Design view (in installation projects) or the Components view, expand the node of the component that contains the file or folder that you want to be removed, and then click the Files subnode. Next, right-click in the Files pane and click Add File Removal.

With both methods, InstallShield displays a Properties dialog box that lets you configure the available removal settings.

Previously, the only way to indicate which files and folders should be removed was to use the Direct Editor view to manually author entries in the **RemoveFile** table.

This feature is available in the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

To learn more, see:

- Removing Files and Folders from Target Systems
- File Removal Properties Dialog Box

The automation interface includes a new ISWiRemoveFile object and a new ISWiRemoveFiles collection for removing files and folders. In addition, three new member methods are applicable to the ISWiComponent object: AddRemoveFile, which enables you to add a file removal entry to the current component; RemoveRemoveFile, which enables you to remove a file removal entry from the current component; and ISWiRemoveFiles, which returns an ISWiRemoveFiles collection that contains all of the file removal entries that are associated with the current component.

These automation interface improvements are available in the following project types: Basic MSI, InstallScript MSI, and Merge Module.

For more details, see:

- ISWiRemoveFile Object
- ISWiRemoveFiles Collection
- AddRemoveFile Method
- RemoveRemoveFile Method
- ISWiRemoveFiles Method
- ISWiComponent Object

Support for DPI-Aware Installations

InstallShield now lets you create DPI-aware installations that can scale run-time user interface elements properly across modern target systems that have various high-DPI display settings, making your installation's user interface more consistent and visually appealing. The support that is available and the improvements that have been made vary, depending on the project type that you are using.

Advanced UI and Suite/Advanced UI Projects

DPI-awareness in Advanced UI and Suite/Advanced UI projects has two parts:

- Items that are drawn by the engine are scaled according to the target system's DPI settings. Thus, if a target system has a DPI of 200%, the check box control is scaled accordingly.

- The engine considers the scale factor and language when displaying images and other resources that you are including in the wizard interface.

For example, when the engine determines that the scale factor should be 150 and that the appropriate language is English, it looks for resources that have the string *scale-150* in their path or file name. It searches the following paths in the order listed when searching for **image.png** on the target system:

1. [SUPPORTDIR]0409**scale-150**\image.png
2. [SUPPORTDIR]0409\image.**scale-150**.png
3. [SUPPORTDIR]0409\image.png
4. [SUPPORTDIR]**scale-150**\image.png
5. [SUPPORTDIR]image.**scale-150**.png
6. [SUPPORTDIR]\image.png

Note that matching the language takes precedence over matching the correct DPI.

InstallShield includes scale-150 and scale-200 images for the built-in default images that are included in Advanced UI and Suite/Advanced UI projects. If you want to include custom resources in your project, use the Support Files view to add appropriately scaled images.

InstallScript and InstallScript MSI Projects

DPI-awareness in InstallScript and InstallScript MSI projects has the following parts:

- All of the built-in default dialog images have been replaced with images that scale well to 200%.
- The static dialog controls that have a control ID of 1200 now include support for .png images, which allow for transparency. This applies to the computer image that is displayed by default on the left side of the exterior skinned dialogs such as the Welcome dialog. It also applies to the banner images on interior dialogs.

  In addition, these same controls support a new format for identifying each image and its associated scale factor:

  @@ResourceID;ScaleFactor

  ResourceID indicates the resource identifier number that should be used to look up either a .png image (stored as resource type PNG) or a bitmap image (stored as resource type RT_BITMAP). *ScaleFactor* indicates the DPI scale percentage for which the image is intended.

  For example, this can be 100% (96 DPI), 125% (120 DPI), 150% (144 DPI), or 200% (192 DPI). If the scale factor that is specified for an image is 200, the image will be shrunk down for display on target systems that are running less than 200% DPI scaling. On a 200% target system, it will be displayed at 1:1. If the scale factor that is specified for the image is 100, the image will be scaled up for display on target systems that are running 200% DPI scaling.

  The previous format for identifying bitmap images (.bmp) still works, but it does not have support for scaling, or for .png images:

  @BitmapResourceID;TransparentFlag;3-DFlag;<unused>;TransparentColorKey
- The new image string format is now used by default on all ID 1200 static controls for the dialog header banner and for exterior panel icon images. If you upgrade an InstallScript or InstallScript MSI project to InstallShield 2016 and you edited the dialogs in the earlier version of InstallShield, you will need to edit the image string to use the new format for all of the languages that your project supports.

- The InstallScript function **SdOptionsButtons** has been updated to include support for the new string format for identifying images:

  @@ResourceID;ScaleFactor

  Previously, the following format was used:

  `@BitmapResourceID;BitmapIconFlag;TransparentColor`

- The nItem parameter for the InstallScript function **GetSystemInfo** includes support for two new constants:
  - **SYSTEM_DPI**—This returns the system DPI value in the nvResult parameter.
  - **SYSTEM_DPI_SCALING**—This returns the system DPI scaling value in the nvResult parameter.

To learn more, see:

- GetSystemInfo
- SdOptionsButtons
- Background About the Default Images on Dialogs in InstallScript and InstallScript MSI Projects

Basic MSI Projects

The built-in images that are displayed on dialogs that the Setup.exe launcher displays at run time have been replaced with images that scale well to larger areas. This includes images that are displayed on the initialization dialog.

Support for Distributing Your Installation to Virtual Machines that InstallShield Provisions at Build Time or on Demand

You can configure your projects so that after each successful build of your installation, InstallShield automatically reverts a virtual machine (VM) to a designated snapshot, powers on the VM, and copies your installation to the VM to make it available for testing. You can also alternately perform these testing preparation steps on demand at any time. The testing preparation capability makes it possible to reduce testing time and eliminate manual steps.

The VM can be on a Microsoft Hyper-V Server, a VMware ESX or ESXi Server, or a VMware Workstation.

To specify VM information, use the Virtual Machine area on the Events tab for a release in the Releases view. One of the settings lets you specify whether you want the release to be distributed to the specified VM each time that it is successfully built.

To distribute the release to the VM at any time, right-click the release in the Releases view and then click Distribute to VM.

Note that when you use the Configuration setting on the Events tab to configure VM details, InstallShield writes the data that you specify to a file called **VMConfigurations.xml**. This **VMConfigurations.xml** file is a machine-wide file that you can configure once and then use it in other InstallShield projects, as well as share it with other team members. You can also use this file with the Standalone Build.

In order to deploy to one of the supported VMs, InstallShield needs to communicate with the virtualization technology that you are using. If you are using VMware virtualization technology (VMware ESX or ESXi Server or VMware Workstation), the VMware VIX API needs to be installed on the same machine as InstallShield. You can do this by either installing VMware Workstation on that machine or by downloading and installing the VMware VIX API from http://www.vmware.com/support/developer/vix-api.

If you are using VMware Workstation, it is recommended that you install VMware Workstation on the same machine as InstallShield so that InstallShield uses the version of the VIX API that was designed for that specific version of VMware Workstation. Although it is likely that newer versions of the VIX API will also work, it seems that the best approach is for InstallShield to use the version of the VIX API that was bundled with your version of VMware Workstation.

This feature is available in the following project types: Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI.

This feature is available in the Premier edition of InstallShield.

To learn more, see the following:

- Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand
- Events Tab for a Release
- Edit Virtual Machine Configurations Dialog Box
- Sharing Virtual Machine Settings for Distribution of Releases

Enhancements

InstallShield includes the following enhancements.

Usability Improvements for Populating Combo Box and List Box Controls in Advanced UI and Suite/Advanced UI Projects

InstallShield lets you add combo box controls and list box controls to wizard pages and secondary windows of Advanced UI and Suite/Advanced UI projects. The process of creating either one of these types of controls involves the use of two properties: one that defines the list of options that you want to display in the control, and one that the installation uses to store the option that the end user selects at run time. Now you can configure both of those properties entirely from within the Wizard Interface view in InstallShield, instead of having to use the Wizard Interface view and the Property Manager view.

In the Wizard Interface view, the Content Property setting of the combo box or list box control now contains an ellipsis button (...) that you can click when you want to define the list of options that you want to display in the control. When you click this button, a simple Edit List Options dialog box opens, enabling you to specify the list of options that you want to display in the selected control, as well as an associated value.

Previously, it was necessary to use the Property Manager view to define the property that contained the list of options, as well as their corresponding property values. The format that was required for defining the property (for example, ID_Option1\rValue1\nID_Option2\rValue2\nID_Option3\rValue3) was difficult to remember and error-prone.

To learn more, see:

- Populating Combo Box and List Box Controls in the Wizard Interface
- Edit List Options Dialog Box

Run-Time Improvements for Browsing to a File or Folder in Advanced UI and Suite/Advanced UI Installations

Several improvements for enabling end users to browse to a file or folder are now available in Advanced UI and Suite/Advanced UI installations:

- A new Browse for New File action is available for controls on wizard pages and secondary windows. If you add this action to the Click event of a control such as a button and an end user clicks the button, a browse dialog box opens, enabling the end user to specify a new file and its location.

  When you add this type of action to a control in your project, you can specify the text for the title bar of the dialog box. You can also specify the file extension filter options that you want to be available for selection, as well as the default file extension that should be used when creating the file.

- The existing Browse for Folder action has been enhanced. The Browse for Folder dialog box that this action launches accepts only locations that have file system equivalents. That is, the OK button on this dialog box remains disabled if the end user selects a location such as This PC or My Computer. Previously, end users were able to select such locations, and doing so cleared the value of the associated property.

  When you add this type of action to a control in your project, you can specify the instructions that you want to be displayed on this Browse for Folder dialog box.

- The existing Browse for File action has been enhanced to enable you to specify the file extension filter options that you want to be available for selection on the file browse dialog box.

For more details, see Configuring an Action for an Element in the Wizard Interface.

Support for Offering Printer Selection for the LicenseAgreement Dialog at Run Time

The behavior of the Print button on LicenseAgreement dialogs has been enhanced. Instead of printing directly to the default printer when an end user clicks the Print button, the printer selection dialog box now opens.

This enhancement is available in the following project types: Basic MSI, InstallScript, InstallScript MSI.

Advanced UI and Suite/Advanced UI projects already have this capability.

Ability to Insert Text in Text Files

The Text File Changes view enables you to configure changes that you want to be made to text files at run time on target systems. The text files can be part of your installation, or they can be files that are already present on target systems. The Text File Changes view has been improved to enable you to insert text at the beginning or end of one or more files. It also now lets you insert text before or after specified text. When you are configuring text file changes in this view, use the new Action setting to specify whether you want to replace existing text, insert text before or after existing text, or insert text at the beginning or end of files.

Previously, the Text File Changes view had support for replacing existing text in text files at run time.

This enhancement is available in the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, Transform.

For more information, see:

- Modifying Text Files
- Text File Changes View

Support for Enabling End Users to Choose to Which Installed Instances of an Earlier Version of a Product to Apply a Major Upgrade

The multi-instance support in Basic MSI projects now includes support for major upgrades. At run time, the instance selection dialog that the **Setup.exe** setup launcher displays lets end users view a list of all of the currently installed instances of the product, including different major versions of the product that are installed. End users can now use this improved instance dialog to install a new instance, maintain an existing instance, or apply a major upgrade for an instance.

Previously, the only instances that the instance selection dialog listed were ones that shared the same product code as the running multi-instance installation.

To learn more, see:

- Configuring and Building a Major Upgrade that Includes Multiple-Instance Support
- Setting Properties for an Instance
- Run-Time Behavior for Installing Multiple Instances of a Product

Ability to Create Microsoft App-V 5.x Packages on Additional Platforms

The Microsoft App-V Assistant in InstallShield includes support for creating virtual applications in the Microsoft App-V 5.x format. The Package Information page in this assistant lets you specify which version of App-V—5.x or 4.x—you want to target. Now you can build an App-V 5.x package in InstallShield on any version of Windows that InstallShield supports. Previously, Windows 8 or Windows Server 2012 was required.

For information, see:

- Components of an App-V Package
- Build Output for App-V Packages

Enhanced File Open Run-Time Dialog for Basic MSI Installations

InstallShield includes support for launching the Open dialog from one of the dialogs in your Basic MSI installation. End users click a browse button in one of your dialogs, and this launches the Open dialog. The Open dialog lets the end user browse for a file. You can now optionally use the new property **IS_BROWSE_INITIALDIR** to specify the default path that is displayed in this dialog at run time. Previously, the dialog displayed the path to which the end user last browsed.

To learn more, see Launching a File Open Dialog.

New FlexNet Connect 13.06 Redistributables Available

InstallShield includes support for FlexNet Connect 13.06 in Basic MSI and InstallScript MSI projects. Use the Update Notifications view in InstallShield to include one of the two FlexNet Connect 13.06 merge modules—one has the Common Software Manager, and the other does not.

What's New in InstallShield 2013 SP1

InstallShield 2013 Service Pack 1 (SP1) includes changes that offer support for the final released versions of Windows 8.1, Windows Server 2012 R2, and Visual Studio 2013.

Ability to Target Windows 8.1 and Windows Server 2012 R2 Systems

InstallShield enables you to specify that your installation requires Windows 8.1 or Windows Server 2012 R2. It also lets you build feature and component conditions for these operating systems.

The InstallShield prerequisites that should be installable on Windows 8.1 and Windows Server 2012 R2 have been updated so that they are installed on those systems if needed. Previously, the prerequisites were not run by default on those systems. This applies to the following InstallShield prerequisites:

- FSharp Redistributable Package 2.0
- JRE_SE 1.7.0_02 (x64)
- JRE_SE 1.7.0_02 (x86)
- Microsoft .NET Framework 3.0 OS Component
- Microsoft .NET Framework 3.5 SP1 (Windows Feature)
- Microsoft .NET Framework 4.5 Full
- Microsoft .NET Framework 4.5 Web
- Microsoft App-V 5.0 SP1 Desktop Client (x64)
- Microsoft App-V 5.0 SP1 Desktop Client (x86)
- Microsoft ReportViewer 2010
- Microsoft SQL CE 3.5 SP2
- Microsoft SQL Server 2005 Express SP3 (x86 & x64Wow)
- Microsoft SQL Server 2005 Express SP3 (x86)
- Microsoft SQL Server 2008 Express SP1 (x64)
- Microsoft SQL Server 2008 Express SP1 (x86 & x64Wow)
- Microsoft SQL Server 2008 Express SP1 (x86)
- Microsoft SQL Server 2008 Management Objects 10.00.2531 (IA64)
- Microsoft SQL Server 2008 Management Objects 10.00.2531 (x64)
- Microsoft SQL Server 2008 Management Objects 10.00.2531 (x86)
- Microsoft SQL Server 2008 Native Client 10.00.2531 (IA64)
- Microsoft SQL Server 2008 Native Client 10.00.2531 (x64)
- Microsoft SQL Server 2008 Native Client 10.00.2531 (x86)
- Microsoft SQL Server 2008 R2 Express RTM (x64)
- Microsoft SQL Server 2008 R2 Express RTM (x86 & x64Wow)
- Microsoft SQL Server 2008 R2 Express RTM (x86)
- Microsoft SQL Server 2008 R2 Express SP2 (x64)
- Microsoft SQL Server 2008 R2 Express SP2 (x86 & x64Wow)
- Microsoft SQL Server 2008 R2 Express SP2 (x86)

- Microsoft SQL Server 2008 R2 Native Client 10.50.1600.1 (IA64)
- Microsoft SQL Server 2008 R2 Native Client 10.50.1600.1 (x64)
- Microsoft SQL Server 2008 R2 Native Client 10.50.1600.1 (x86)
- Microsoft SQL Server 2012 Express LocalDB RTM (x64)
- Microsoft SQL Server 2012 Express LocalDB RTM (x86)
- Microsoft SQL Server 2012 Express RTM (x64)
- Microsoft SQL Server 2012 Express RTM (x86 & x64Wow)
- Microsoft SQL Server 2012 Express RTM (x86)
- Microsoft SQL Server 2012 Native Client (x64)
- Microsoft SQL Server 2012 Native Client (x86)
- Microsoft SQL Server Compact 4.0 (x64)
- Microsoft SQL Server Compact 4.0 (x86)
- Microsoft SQL Server Native Client 9.00.4035 (IA64)
- Microsoft SQL Server Native Client 9.00.4035 (x64)
- Microsoft SQL Server Native Client 9.00.4035 (x86)
- Microsoft SQL Server System CLR Types 10.00.2531 (IA64)
- Microsoft SQL Server System CLR Types 10.00.2531 (x64)
- Microsoft SQL Server System CLR Types 10.00.2531 (x86)
- Microsoft Visual C++ 2005 SP1 Redistributable MFC Security Update KB2538242(x64)
- Microsoft Visual C++ 2005 SP1 Redistributable MFC Security Update KB2538242(x86)
- Microsoft Visual C++ 2005 SP1 Redistributable Package (x64)
- Microsoft Visual C++ 2005 SP1 Redistributable Package (x86)
- Microsoft Visual C++ 2008 SP1 Redistributable MFC Security Update KB2538243(x64)
- Microsoft Visual C++ 2008 SP1 Redistributable MFC Security Update KB2538243(x86)
- Microsoft Visual C++ 2008 SP1 Redistributable Package (x64)
- Microsoft Visual C++ 2008 SP1 Redistributable Package (x86)
- Microsoft Visual C++ 2010 Redistributable Package (x64)
- Microsoft Visual C++ 2010 Redistributable Package (x86)
- Microsoft Visual C++ 2010 RTM Redistributable MFC Security Update KB2467173 (x64)
- Microsoft Visual C++ 2010 RTM Redistributable MFC Security Update KB2467173 (x86)
- Microsoft Visual C++ 2010 SP1 Redistributable Package (x64)
- Microsoft Visual C++ 2010 SP1 Redistributable Package (x86)
- Microsoft Visual C++ 2012 Redistributable Package (x64)

- Microsoft Visual C++ 2012 Redistributable Package (x86)
- Microsoft Visual C++ 2012 Update 1 Redistributable Package (x64)
- Microsoft Visual C++ 2012 Update 1 Redistributable Package (x86)
- Microsoft VSTO 2010 Runtime (x64)
- Microsoft VSTO 2010 Runtime

Enhancements to the InstallScript Language for Windows 8.1 and Windows Server 2012 R2

The following structure members and predefined constants were added to the InstallScript language:

- **SYSINFO.WINNT.bWin81**—This is a new SYSINFO structure member. If the operating system is Windows 8.1 or Windows Server 2012 R2, this value is TRUE.
- **ISOSL_WIN81**—This is a new predefined constant that is available for use with the **FeatureFilterOS** function and the SYSINFO structure variable. It indicates that the target system is running Windows 8.1 or Windows Server 2012 R2.

For more information, see FeatureFilterOS function and the SYSINFO structure variable.

Automation Interface Enhancement: OSFilter Property Value for Windows 8.1 and Windows Server 2012 R2

The following constants are now available for use with the OSFilter member of the ISWiComponent and ISWiRelease objects in the automation interface:

eosWin81 = &H8000000 (134217728)—These are for Windows 8.1 and Windows Server 2012 R2.

In addition, the value for the eosAll constant is now &HFD100D0 (265355472); previously, it was &7D100D0 (131137744).

The OSFilter member applies to the ISWiComponent object in InstallScript and InstallScript MSI projects. The OSFilter member applies to the ISWiRelease object in InstallScript projects.

For more information, see the following:

- ISWiComponent Object
- ISWiRelease Object

Support for Microsoft Visual Studio 2013

InstallShield includes support for Visual Studio 2013. You can create InstallShield projects from within this version of Visual Studio.

New InstallShield Prerequisites for Microsoft SQL Server 2012 Express SP1

InstallShield includes several new SQL Server–related InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Microsoft SQL Server 2012 Express SP1 LocalDB (x64)
- Microsoft SQL Server 2012 Express SP1 LocalDB (x86)

- Microsoft SQL Server 2012 Express SP1 (x64)
- Microsoft SQL Server 2012 Express SP1 (x86 & x64Wow)
- Microsoft SQL Server 2012 Express SP1 (x86)
- Microsoft SQL Server 2012 Express SP1 Management Objects (x64)
- Microsoft SQL Server 2012 Express SP1 Management Objects (x86)
- Microsoft SQL Server 2012 Express SP1 System CLR Types (x64)
- Microsoft SQL Server 2012 Express SP1 System CLR Types (x86)

These InstallShield prerequisites install the technology on supported target systems.

Support for Defining Custom Themes for the Controls on Wizard Pages and Windows in Suite/Advanced UI and Advanced UI Installations

Suite/Advanced UI and Advanced UI projects now enable you to define themes for wizard interface controls. A control theme is a collection of colors for various parts of controls—text color, background color, and border color—and for various states of controls—such as clicked and hovered. You can specify the control theme that you want to use for the controls in your user interface by default. You can also override the theme for specific controls on wizard pages and windows in your user interface. Control themes give buttons and other items in your installation's user interface a flat look (without the three-dimensional effects) that is reminiscent of the style of Windows Store apps.

The following types of controls support the use of control themes:

- Buttons, including the navigation buttons
- Check boxes
- Radio buttons
- Command links

To add a control theme to your project, right-click the Control Themes node in the Wizard Interface view, and then click Add Control Theme. Then configure the theme's settings as needed.

To specify the theme that you want to use by default for the controls in your user interface, select the appropriate theme in the new Default Control Theme setting that is displayed in the Wizard Interface view when the Wizard Pages node is selected. This setting is blank by default.

To override the control theme for a specific control, select the appropriate theme in the Control Theme setting for that control.

For details, see:

- Using Styles, Brushes, and Control Themes to Customize the Wizard Interface
- Defining a Custom Control Theme for the Wizard Interface
- Applying a Theme to a Control in the Wizard Interface

New Predefined Path Variable for the Visual Studio Solution Folder

A new predefined path variable called VSSolutionFolder is available in projects to reference a higher-level base directory. This support enables you to have in your InstallShield projects static links to files in sibling projects that are within the Visual Studio solution folder; if you work on the projects on a different machine, the static links that use the VSSolutionFolder path variable can reference the correct paths for the files in sibling projects.

The VSSolutionFolder path variable is defined automatically whenever an InstallShield project is opened from within a Visual Studio solution. It is also defined automatically if you are using MSBuild to build a solution that contains an InstallShield project. However, in other scenarios, when the InstallShield project is opened without the Visual Studio solution, VSSolutionFolder cannot be defined automatically. For example, if you open the InstallShield project in InstallShield directly, without having Visual Studio open, VSSolutionFolder is not defined. Similarly, if you use the command-line tool **IsCmdBld.exe**, or if you use MSBuild with an .isproj file, VSSolutionFolder is not defined. If you are using **IsCmdBld.exe** to build a release in InstallShield project, use the `-L` command-line parameter to set the value of VSSolutionFolder. If you are using MSBuild, use the PathVariables parameter to set the value of VSSolutionFolder. This parameter is exposed as the ItemGroup InstallShieldPathVariableOverrides when the default targets file is used.

If you include in your InstallShield project a source file whose path includes the VSSolutionFolder path variable and build it in an environment that does not support the VSSolutionFolder path variable, build errors such as the following ones may occur:

- -6103: Could not find file <VSSolutionFolder>\MyFile.exe
- -6271: File <VSSolutionFolder>\MyFile.exe not found. An error occurred building the MsiFileHash table record for this file. Verify that the file exists in the specified location.

This functionality is available in the following project types: Advanced UI, Basic MSI, InstallScript, InstallScript MSI, Merge Module, Suite/Advanced UI.

To learn more, see:

- Using the VSSolutionFolder Path Variable with Visual Studio Solutions
- Predefined Path Variables
- Microsoft Build Engine (MSBuild)
- ISCmdBld.exe

Ability to Easily Update the Product Codes of Instances in a Multi-Instance Project

If you are creating major upgrades for a project that includes multi-instance support, it is necessary to update the product code of each instance. InstallShield now enables you to easily update the product code of a specific instance that is defined in your project. InstallShield also now lets you easily update the product code of all instances that are defined for a product configuration in your project.

To update the product code of a specific instance, use the new Generate a new GUID command that is available when you right-click the value in the ProductCode setting for an instance that is defined on the Multiple Instances tab for a product configuration in the Releases view.

To update the product code of each instance that is defined for a product configuration in your project, use the new Change ProductCode of Each Instance command that is available when you right-click the Instances node on the Multiple Instances tab for a product configuration in the Releases view.

This functionality is available in Basic MSI projects.

For details, see Updating the Product Code of One or More Instances.

What's New in InstallShield 2013

New Features

InstallShield includes the following new features.

Ability to Create Run-Time Actions That Extend the Behavior of a Suite/Advanced UI Installation

You may require your Suite/Advanced UI installation to perform various run-time tasks that are outside the scope of the packages that you are including in the installation. For example, you may need the Suite/Advanced UI installation to do one or more of the following:

- Install an application before the user interface of the Suite/Advanced UI installation is displayed.

  One sample scenario in which this may be necessary is when one of the packages in your Suite/Advanced UI installation runs SQL scripts to install an Oracle database. Before this package is run, you may want the Suite/Advanced UI interface to display a wizard page that lets end users select the appropriate server from a list of servers that are available on the network. This type of UI support requires that ODBC drivers be installed on the target system.

- Search the target system for the presence or absence of a particular product, technology, folder, file, registry entry, or other item.

- Configure the target system before or after running a package in the Suite/Advanced UI installation.

To extend the capabilities of a Suite/Advanced UI installation and make it possible to perform those sorts of tasks and more, you can use the new Events view in your Suite/Advanced UI project to create actions that run executable files, call DLL functions, run PowerShell scripts, or set Suite/Advanced UI properties at run time.

When you add an action to your project, you can schedule when at run time you want it be launched:

- Use the Events explorer in the Events view to schedule the action to run during one or more of the built-in events that the Suite/Advanced UI manages. This area of the view lists the events and the actions that occur during each event in chronological order from top to bottom.

- Use the settings in the Events area of the Packages view to schedule an action to run before or after the Suite/Advanced UI engine installs, removes, modifies, or repairs a package.

When you schedule an action, you can build conditional statements that control whether the action should be run. You can use the same types of condition checks that are available in other areas of a Suite/Advanced UI project, as well as two additional new types of condition checks:

- **Package Operation**—Check the state (for example, install or remove) in which a particular package in the Suite/Advanced UI installation is running.

- **Feature Operation**—Check the state (for example, install or remove) in which a particular feature in the Suite/Advanced UI installation is running.

These types of conditions are available only for actions that are associated with an event in the Events view, or with a package in the Packages view.

Note that the PowerShell action support requires PowerShell 2.0 or later on target systems.

This new feature is available in the Premier edition of InstallShield.

For more information, see:

- Using Actions to Extend the Behavior of a Suite/Advanced UI Installation
- Working with an .exe File for an Action in a Suite/Advanced UI Installation
- Working with a DLL File for an Action in a Suite/Advanced UI Installation
- Working with a PowerShell Script for an Action in a Suite/Advanced UI Installation
- Adding an Action to a Suite/Advanced UI Project
- Configuring a Suite/Advanced UI Action's Settings
- Scheduling a Suite/Advanced UI Action
- Types of Events in a Suite/Advanced UI Installation
- Events View
- Packages View
- Package Operation Condition Settings
- Feature Operation Condition Settings

Support for Enabling Windows Roles and Features During a Suite/Advanced UI Installation

If a particular package in your Suite/Advanced UI installation requires that one or more Windows roles and features be enabled on target systems, you can specify this requirement through the new Windows Features setting in the Packages view when you are configuring the package in your Suite/Advanced UI project. At run time, if an eligible package that is being installed requires one or more Windows roles or features that are disabled, the Suite/Advanced UI installation enables those roles and features before launching the package.

For example, your Suite/Advanced UI project may have a package that installs an IIS Web site that requires that the Internet Information Services feature in Windows be turned on. Another package in the same project may require that the PowerShell feature be turned on. When you are configuring the settings of those packages in your project, you can specify the Windows features that are required; at run time, if a package is eligible to be installed on a target system but any of its required Windows features are disabled, the Suite/Advanced UI installation enables those disabled Windows features before launching it. If the package is not eligible, its required Windows features are not enabled.

InstallShield has built-in support for enabling several Windows features:

- Internet Information Services
- PowerShell
- .NET Framework 3.x

InstallShield also lets you specify additional Windows roles and features that a package requires.

This new feature is available in the Premier edition of InstallShield.

To learn more, see:

- Enabling Windows Roles and Features During a Suite/Advanced UI Installation
- Packages View

Ability to Create Advanced UI and Suite/Advanced UI Project Templates

InstallShield has support for using Advanced UI and Suite/Advanced UI projects as templates. A project template contains default settings and design elements that you want to use as a starting point when you create a new Advanced UI or Suite/Advanced UI project.

You can designate any Advanced UI or Suite Advanced UI project to be a template. Project files and template files have the same file extension: .issuite.

To designate that an .issuite project file is a template file and make the template available for selection in the New Project dialog box, right-click in the All Types tab on the New Project dialog box and then click Add New Template. InstallShield lets you browse to the .issuite file that you want to use as a template, and it adds a new icon for it to the All Types tab.

To create a new Advanced UI or Suite/Advanced UI project using your .issuite file as a template, select the template's icon on the All Types tab of the New Project dialog box when you are creating the new project.

For instructions, see:

- Creating Project Templates
- Basing New Projects on Templates

New Built-in Template Available for Creating Installations that Install Multi-tier Applications

InstallShield includes a new Multi-tier Application template that helps you get started with creating a Suite/Advanced UI installation for cloud-based multi-tier applications. You can create this sort of installation to enable end users to easily manage the installation of tiers that install Web sites, databases, and applications. They can use the wizard pages in the Suite/Advanced UI installation to select which tiers they want to install, or they can use the command-line support to select the appropriate features.

The new Multi-tier Application template contains three tiers that are exposed as features. You can customize these features to include the tiers of your product, adding or removing features as needed. The template also contains four releases in the Releases view: one flagged for each of the three predefined features, and one that includes all of the features. This enables you to build separate installations—one for each tier, and one that lets end users install all available tiers.

This feature is available in the Premier edition of InstallShield.

New InstallShield Prerequisites for .NET Framework 3.5 SP1, Microsoft Visual C++ 2012, and SQL Server 2008 R2 Express SP2

InstallShield includes the following InstallShield prerequisites that you can add to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects:

- Microsoft .NET Framework 3.5 SP1 (Windows Feature)
- Microsoft SQL Server 2008 R2 Express SP2 (x64)
- Microsoft SQL Server 2008 R2 Express SP2 (x86 & x64Wow)
- Microsoft SQL Server 2008 R2 Express SP2 (x86)
- Microsoft Visual C++ 2012 Update 1 Redistributable Package (x64)
- Microsoft Visual C++ 2012 Update 1 Redistributable Package (x86)

These prerequisites install the various technologies on supported target systems.

New Application Virtualization Suitability Tests

Several new validation suites are available in InstallShield for helping you to determine how ready your products are for virtualization. The InstallShield virtualization internal consistency evaluators (ISVICEs) that are included in these suites let you check suitability for Microsoft App-V, Microsoft Server App-V, VMware ThinApp, and Citrix XenApp. The validation suites can enable you to make more informed decisions about how you build your product if you are considering offering your customers a virtualized version.

If you want to configure InstallShield to perform validation with these validation suites each time that a Basic MSI release is successfully built: On the Tools menu, click Options. On the Validation tab, select the appropriate check boxes.

If you want to perform validation separately from the build process: On the Build menu, point to Validation, and then click the appropriate new suite.

The virtualization suites are available in the Premier Edition of InstallShield.

This feature is available in the following project types: Basic MSI and InstallScript MSI.

For more information, see:

- InstallShield Virtualization Suitability Suite
- Validating Projects

Support for Creating Microsoft App-V 5.x Packages

The Microsoft App-V Assistant in InstallShield includes support for creating virtual applications in the Microsoft App-V 5.x format. The Package Information page in this assistant lets you specify which version of App-V—5.x or 4.x—you want to target.

If you are targeting App-V 5.x, a new version of the App-V Launcher tool is available for testing App-V 5.x packages before moving them to the deployment server. The new version of this tool is capable of launching a Command Prompt window within the virtual environment of the App-V package. Thus, it is no longer necessary to inject diagnostic tool shortcuts for **Cmd.exe** or **Regedit.exe** directly into an App-V package beginning with App-V 5.x.

The name of the directory in which InstallShield saves an App-V package at build time varies, depending on which version of App-V you are targeting. For App-V 5.x packages, the directory is ProductName, and it is placed in a folder called App-VPackage. For App-V 4.x packages, InstallShield includes the product version number to the ProductName folder; that is, ProductName_v*N*, (where *N* is the product version number).

Note that although you can configure settings for an App-V 5.x package on any version of Windows that InstallShield supports, building an App-V 5.x package from within InstallShield requires Windows 8 or Windows Server 2012. Attempting to build an App-V 5.x package on an earlier version of Windows causes build error -9424.

The Microsoft App-V Assistant is available in the Premier Edition of InstallShield.

For information, see:

- Components of an App-V Package
- Build Output for App-V Packages

New Property Comparison Type of Condition Check; New Property for Determining the Install Mode for Suite/Advanced UI and Advanced UI Projects

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in a Suite/Advanced UI or Advanced UI project, or for an action in a Suite/Advanced UI project, you can select from a number of different types of checks that you want to be evaluated on target systems. Use the new Property Comparison type of condition check to check the value of a particular built-in Advanced UI or Suite/Advanced UI property, or a property that is defined in the Property Manager view.

In addition, a new read-only property called **ISInstallMode** is now available. This property stores a value that indicates the mode (first-time installation, maintenance/UI maintenance selection, modify, remove, repair, or stage only) in which the Suite/Advanced UI or Advanced UI installation is running. You can use this property in conditional statements that you configure in your project.

For more information, see:

- Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects
- Property Comparison Condition Settings
- Advanced UI and Suite/Advanced UI Property Reference

Ability to Build Pure 64-Bit .msi and .msm Packages; New Build-Time Architecture Validation

Windows Server Core supports disabling 32-bit Windows-on-Windows (WOW64) support. As this configuration becomes more popular, you may want to ensure that your 64-bit applications can install without any reliance on 32-bit functionality. To make this possible, InstallShield now enables you to build pure 64-bit .msi packages; these can be run on 64-bit Windows-based systems that do not have WOW64 functionality. You can also build pure 64-bit merge modules and include them in InstallShield projects that have 64-bit support. If your installation or merge module project targets a pure 64-bit environment and includes support that requires any built-in InstallShield custom action DLLs, InstallShield includes new 64-bit versions of these DLLs in your releases at build time.

InstallShield now has architecture validation that you can use when building a release in InstallShield. Architecture validation enables you to detect potentially problematic cases in which your installation may try to install product files or use run-time binaries that may not match the architecture of a target system.

For example, if end users may run your installation on 64-bit target systems that do not have WOW64 support, architecture validation can help you identify any 32-bit product files or 32-bit custom action files in your installation. In addition, if you are mixing 64-bit and 32-bit product files (or 64-bit and 32-bit custom actions) in a single project and generating separate 32-bit and 64-bit .msi packages, you can use architecture validation to identify any 64-bit product or custom action files in your 32-bit releases, since these files cannot be loaded on 32-bit target systems.

To specify what type of architecture validation you want to use and identify whether you want to build a pure 32-bit or 64-bit package, use the new Architecture Validation setting, which is available on the General tab when you select a product configuration in the Releases view. Two types of validation are available: strict and lenient.

Strict validation may trigger build errors if the architecture that the Template Summary property specifies does not match the architecture for one or more of the custom action files that are being included in the release. This type of validation may also trigger build warnings if the architecture that the Template Summary property specifies does not match the architecture for one or more of the product files that are being included in the release.

Lenient architecture validation, which is the default type of validation, does not trigger build errors or warnings if the architecture that the Template Summary property specifies does not match the architecture for one or more of the product files or the custom action files that are being included in the release.

The automation interface includes support for the new architecture validation support. The ISWiProductConfig object includes a new read-write ArchitectureValidation property that lets you specify the type of architecture validation that you want to use for a product configuration.

This feature is available in the following project types: Basic MSI and Merge Module.

To learn more, see:

- Selecting the Appropriate Type of Architecture Validation for Builds
- Targeting 64-Bit Operating Systems
- Using the Template Summary Property
- ISWiProductConfig Object

Validation for the Windows 8 Logo Program

InstallShield includes two new validation suites: InstallShield Validation Suite for Windows 8 and InstallShield Merge Module Validation Suite for Windows 8. These validation suites contain several new InstallShield internal consistency evaluators (ISICEs): ISICE21 through ISICE26. The suites can help you verify whether your Windows Installer–based installation or merge module meets the installation requirements of the Windows 8 Desktop App Certification Program. If you want to be able to use the Windows 8 logo artwork, your product's installation must meet the program's requirements. These suites can also help you verify that your installation or merge module will work on Windows Server 2012 systems.

If you want to configure InstallShield to perform validation with these validation suites each time that a release is successfully built: On the Tools menu, click Options. On the Validation tab, select the appropriate check boxes.

If you want to perform validation separately from the build process: On the Build menu, point to Validation, and then click the appropriate new suite.

For more information, see the following:

- Validating Projects
- ISICEs
- Requirements for the Windows Logo Program

Improvements for Configuring Shortcuts on the Latest Platforms; InstallScript Language Improvements for Shortcut Creation

InstallShield offers enhancements for configuring shortcuts in your projects.

Support for Preventing a Shortcut from Being Pinned to the Windows 8 Start Screen

InstallShield lets you specify whether you want each shortcut in your installation to be pinned by default to the Start screen on Windows 8 target systems. You may want to disable pinning for shortcuts that are for tools and secondary products that are part of your installation. If you disable pinning for a shortcut, the shortcut is still available in the Apps list that contains shortcuts to all of the applications on the system.

To prevent Start Screen pinning for a shortcut, use the new Pin to Windows 8 Start Screen setting for a shortcut in the Shortcuts view.

This feature is available for the following project types: Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

The automation interface includes support for specifying whether you want a shortcut to be pinned to the Windows 8 Start screen. The ISWiShortcut object includes a new read-write Boolean EnableWin8StartPin property that indicates whether the shortcut is configured to be pinned by default to the Start screen on Windows 8 target systems.

To learn more, see:

- Suppressing Initial Pinning of a Shortcut to the Windows 8 Start Screen
- Shortcuts View
- ISWiShortcut Object

Support for Preventing End Users from Pinning a Shortcut to the Taskbar or Start Menu

For each shortcut in your installation, InstallShield lets you specify whether you want to let end users pin the shortcut to the taskbar or to the Start menu. If you want to prevent this type of pinning, the installation sets a property that hides the context menu commands for pinning the shortcut to the taskbar and the Start menu. You may want to prevent pinning for shortcuts that are for tools and secondary products that are part of your installation.

Support for preventing end users from pinning a shortcut to the taskbar or Start menu varies, depending on the project type that you are using.

In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects: To prevent taskbar and Start menu pinning for a shortcut, use the Shell Properties setting for a shortcut in the Shortcuts view. This setting has a new Prevent Pinning option that lets you disable pinning for the selected shortcut. Windows Installer 5 includes support for this shortcut setting. Previously, it was necessary to manually enter the appropriate property name and value to configure this behavior.

In InstallScript projects: To prevent taskbar and Start menu pinning for a shortcut, use the new Prevent Pinning setting in the Shortcuts view. Windows 7 and later include support for this shortcut setting. Previously, InstallShield did not have support for configuring this functionality in InstallScript projects.

The automation interface includes support for specifying whether you want the context menu commands for pinning a shortcut to the taskbar and to the Start menu to be displayed after end users install your product. The ISWiShortcut object includes a new read-write Boolean PreventPinning property that indicates whether the shortcut is configured to hide the context menu commands for pinning.

For more information, see:

- Specifying Whether End Users Should Be Able to Pin a Shortcut to the Taskbar or Start Menu
- Shortcuts View
- ISWiShortcut Object

Support for Preventing a Shortcut on the Start Menu from Being Highlighted as Newly Installed

InstallShield lets you optionally prevent a shortcut on the Start Menu from being highlighted as newly installed after end users install your product. This has the same effect as clearing the **Highlight newly installed programs** check box in the Customize Start Menu dialog box for an individual item on a target system. You may want to set this property for shortcuts that are for tools and secondary products that are part of your installation.

Support for the highlighting behavior varies, depending on the project type that you are using.

In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects: To prevent the highlighting behavior for a shortcut, use the Shell Properties setting for a shortcut in the Shortcuts view. This setting has a new Do Not Highlight as New option that lets you disable highlighting for the selected shortcut. Windows Installer 5 includes support for this shortcut setting. Previously, it was necessary to manually enter the appropriate property name and value to configure this behavior.

In InstallScript projects: To prevent the highlighting behavior for a shortcut, use the new Do Not Highlight as New setting in the Shortcuts view. Windows 7 and later include support for this shortcut setting. Previously, InstallShield did not have support for configuring this functionality in InstallScript projects.

The automation interface includes support for specifying whether you want a shortcut to be highlighted on the Start menu as new after end users install the product. The ISWiShortcut object includes a new read-write Boolean DoNotHighlightAsNew property that indicates whether you want to prevent the highlighting.

To learn more, see:

- Preventing a Shortcut on the Start Menu from Being Highlighted as Newly Installed
- Shortcuts View
- ISWiShortcut Object

Built-in Support for Configuring Additional Windows Shell Properties for a Shortcut

InstallShield lets you specify one or more additional shortcut properties that need to be set by the Windows Shell at run time. The properties that the Shell can set are defined in **propkey.h**, which is part of the Windows SDK.

To configure additional properties for a shortcut, use the Shell Properties setting for a shortcut in the Shortcuts view. This setting has a new Custom Property option that lets you specify additional properties and their corresponding values for the selected shortcut.

Windows Installer 5 includes support for configuring Shell properties.

The automation interface includes a new ISWiShellProperty object for custom Windows Shell properties for a shortcut. In addition, the ISWiShortcut object includes two new methods and a collection that let you add a Shell property (AddShellProperty), delete a Shell property (DeleteShellProperty), and retrieve the collection of Shell properties (ISWiShellProperties) for a shortcut.

This support is available in the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

To learn more, see:

- Setting Custom Shell Properties
- Shortcuts View
- ISWiShellProperty Object
- AddShellProperty Method
- DeleteShellProperty Method
- ISWiShellProperties Collection
- ISWiShortcut Object

InstallScript Language Improvements for Setting a Shortcut Property, and for Other Shortcut Functionality

A new InstallScript function called **SetShortcutProperty** is available. You can use this function in your InstallScript code to set Windows Shell properties for a shortcut. The function lets you prevent end users from pinning a shortcut to the Windows 8 Start screen, prevent end users from pinning a shortcut to the taskbar or Start menu on Windows 7 or later systems, or prevent a shortcut on the Start menu from being highlighted as newly installed on Windows 7 or later systems. You can also use this function to specify additional Shell properties that are defined in **propkey.h**, which is part of the Windows SDK.

The InstallShield Cabinet and Log File Viewer has been updated to reflect the state of the aforementioned Shell properties.

To reflect the current operating system terminology, some of the existing shortcut functions have been renamed. Note that the old function names will continue to compile and run as they did with earlier versions of InstallShield. The new InstallScript functions are:

- **CreateShortcut**—This function supersedes AddFolderIcon. **CreateShortcut** accepts the same parameters as **AddFolderIcon** but adds or replaces the following options for the nFlag parameter:
  - CS_OPTION_FLAG_REPLACE_EXISTING
  - CS_OPTION_FLAG_RUN_MAXIMIZED
  - CS_OPTION_FLAG_RUN_MINIMIZED
  - CS_OPTION_FLAG_PREVENT_PINNING
  - CS_OPTION_FLAG_NO_NEW_INSTALL_HIGHLIGHT
  - CS_OPTION_FLAG_NO_STARTSCREEN_PIN
- **CreateShortcutFolder**—This function supersedes **CreateProgramFolder**. Both functions use the same parameters.
- **DeleteShortcut**—This function supersedes **DeleteFolderIcon**. Both functions use the same parameters.
- **DeleteShortcutFolder**—This function supersedes **DeleteProgramFolder**. Both functions use the same parameters.
- **GetShortcutInfo**—This function supersedes **QueryProgItem**. Both functions use the same parameters.
- **ReplaceShortcut**—This function supersedes **ReplaceFolderIcon**. **ReplaceShortcut** accepts the same parameters as **ReplaceFolderIcon** but adds or replaces the same options for the nFlag parameter as **CreateShortcut**.

For more details, see:

- CreateShortcut
- CreateShortcut Examples
- CreateShortcutFolder
- CreateShortcutFolder Example
- DeleteShortcut
- DeleteShortcut Example
- DeleteShortcutFolder

- DeleteShortcutFolder Example
- GetShortcutInfo
- GetShortcutInfo Example
- ReplaceShortcut
- ReplaceShortcut Example
- SetShortcutProperty
- SetShortcutProperty Example

Usability Improvements for Customizing the Wizard Interface in Advanced UI and Suite/Advanced UI Projects

No More Confusing Syntax Requirements

Various UI settings in the Wizard Interface view in Advanced UI and Suite/Advanced UI projects have been improved to make it much easier to customize the wizard interface of Advanced UI and Suite/Advanced UI installations. For example, the following settings, which are displayed when a wizard page or wizard control is selected in the Wizard Interface view, have been improved:

- **Visible**—This setting lets you specify one or more conditions that the Advanced UI or Suite/Advanced UI installation should use to evaluate whether the selected page or control should be displayed.
- **Enabled**—This setting lets you specify one or more conditions that the Advanced UI or Suite/Advanced UI installation should use to evaluate whether the selected control should be enabled.
- **Validate**—This setting lets you specify one or more validation statements that the Advanced UI or Suite/Advanced UI installation should use to validate an end user's response to a control. For example, you can use this setting to specify the format that end users should use for entering a serial number in a text box control.
- **Action**—Each instance of this setting has been renamed to better reflect the specific type of trigger that is applicable to the selected control. For example, the Action setting for a list box control has been renamed Selection Changed. The Action setting for a button has been renamed Click. The renamed settings let you select one or more actions that you want to be triggered when an end user uses the selected control.

Now those settings contain drop-down lists and buttons that are similar to the ones that are available in other areas of Advanced UI and Suite/Advanced UI projects for quickly building conditional statements. These settings now make it easy to quickly configure visible/hidden status, enabled/disabled status, validation, and action responses for various parts of the UI, and to build conditional statements for that behavior if appropriate. Previously, these settings were edit fields that required manually entry of statements using a difficult-to-remember syntax.

For more information, see:

- Configuring Validation for a Control on a Wizard Page or Window
- Configuring an Action for an Element in the Wizard Interface
- Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects

Built-In Support for Using Only One Background for the Header, Body, and Navigation Areas of the Wizard Interface

The Wizard Pages node in the Wizard Interface view has a new Full Wizard Background setting. This setting lets you specify a single background that you want to use across the header, body, and navigation areas of your wizard pages. Previously, the only built-in way to specify backgrounds in InstallShield was to use separate background

styles in the Default Body Background, Header Background, and Navigation Background settings. If you wanted to use a single background across all three areas of the wizard interface, you had to use the process that was documented in an InstallTalk blog article; the process involved modifying the InstallShield project file (.issuite) in a text editor.

Note that if you select a background in this new Full Wizard Background setting, you should delete any values from the other background settings: Default Body Background, Header Background, and Navigation Background.

To learn more, see Specifying the Background Brush for the Header, Body, and Navigation Areas of the Wizard Interface.

Ability to Specify a Global Text Style for All Navigation Buttons

The Wizard Pages node in the Wizard Interface view has a new Navigation Text Style setting. This setting lets you select a single text style that you want to use for the text on all of the navigation buttons. You can override the global text style for individual navigation buttons as needed.

Logically Organized Setting Grids

In the Wizard Interface view, the setting grids for wizard pages, secondary windows, and wizard controls have been reorganized to make it easier to find the settings that are needed for customizing the wizard interface.

Support for Defining Font Sets with Optional Language-Specific Choices for Advanced UI and Suite/Advanced UI Projects

Advanced UI and Suite/Advanced UI projects have support for a new kind of wizard interface style: a font set. A font set is a collection of fonts (including attributes such as font name, size, and weight). For each font in a font set, you can specify to which language the font is applicable. This enables you to select a different font for each language that your project supports.

Font sets work in conjunction with text styles. Text styles, which are styles that define text attributes such as text color and alignment, now reference a font set but can optionally override various font attributes that are defined at the font set level.

Thus, for example, when you are specifying font information for the body text of your wizard interface, you can use a font set called BodyFonts, and specify a different font for each of the languages that your project supports. You can then select the BodyFonts font set for various text styles—Body, Header, and BodyBold—but with special overrides for smaller sizes and bold where necessary. This enables you to specify a large set of possible fonts once, and change attributes such as text size or color for specific wizard interface uses.

To learn more, see:

- Using Styles, Brushes, and Control Themes to Customize the Wizard Interface
- Defining a Custom Style for the Wizard Interface

Enhancements

InstallShield includes the following enhancements.

New Services View

InstallShield has a new Services view—under the System Configuration node of the View List—that lets you specify the required information about a Windows service that you are installing, starting, configuring, stopping, or deleting at run time.

This new Services view has been added to make it easier for new users to configure services. The view provides the same functionality as the following existing areas:

- The Services area under the Advanced Settings node for a component in the Setup Design view (of installation projects)
- The Services area under the Advanced Settings node for a component in the Components view

If you configure service information in any of the three views (the Services view, the Setup Design view, or the Components view), the other views are updated automatically.

The Services view is available in the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

For more information, see:

- Installing, Controlling, and Configuring Windows Services
- Services View

New Wizard Format for the Suite/Advanced UI and Advanced UI Wizard Interface

A new Glass wizard format is available for the wizard interface of Suite/Advanced UI and Advanced UI installations. This new format is similar to the existing Aero format. For both formats, the installation uses user-chosen system colors (instead of brush styles that you defined in your project) to display the caption bar, header, and navigation areas of the wizard interface. It also may use the glass effect (translucency) for these same areas of the wizard interface on target systems that have the translucency support that was introduced in Windows Vista.

In some scenarios, an installation uses the Wizard 97 format instead of the Glass or Aero formats:

- Windows 8 and Windows Server 2012 systems do not have translucency support. On these systems, a Glass-formatted wizard interface uses the Wizard 97 format. However, an Aero-formatted wizard interface uses Aero (which includes user-chosen system colors), but without the glass effect.
- Windows 7, Windows Vista, Windows Server 2008 R2, and Windows Server 2008 have translucency support that end users can enable or disable. On these systems, Glass-formatted wizard interfaces and Aero-formatted wizard interfaces use the Wizard 97 format if translucency is disabled. These wizard interfaces also use the Wizard 97 format if the desktop composition feature on the target system is disabled.
- On Windows XP and Windows Server 2003, Glass-formatted wizard interfaces and Aero-formatted wizard interfaces use the Wizard 97 format.

To configure the format, use the Wizard Format setting that is displayed in the Wizard Interface view when the Wizard Pages node is selected. The Glass format is the default option for all new Suite/Advanced UI and Advanced UI projects.

For more information, see Selecting the Format for the Wizard Interface.

Automation Interface Enhancement: New OnUpgrade Property for Minor Upgrades and Small Updates

A new read-write OnUpgrade property has been added to the ISWiProject object in the automation interface. You can set this property to one of the supported values to specify whether you want the installation to display a prompt before installing a minor upgrade or a small update.

This property corresponds with the Small/Minor Upgrade Settings options on the Common tab in the Upgrades view.

For more information, see ISWiProject Object.

What's New in InstallShield 2012 Spring SP1

InstallShield 2012 Spring Service Pack 1 (SP1) includes changes that offer support for the final released versions of Windows 8, Windows Server 2012, and Visual Studio 2012. It also includes additional changes.



Important • *If you want to open an InstallShield 2012 Spring Advanced UI or Suite/Advanced UI project (.issuite) in InstallShield 2012 Spring SP1, you must allow InstallShield to upgrade your project to InstallShield 2012 Spring SP1. InstallShield 2012 Spring SP1 Advanced UI and Suite/Advanced UI projects include support for functionality that was not available in these project types in InstallShield 2012 Spring, and this support needs to be added during the upgrade. Note that it is not possible to open InstallShield 2012 Spring SP1 Advanced UI or Suite/Advanced UI projects in earlier versions of InstallShield (including InstallShield 2012 Spring without SP1). Therefore, if multiple users need to open and modify your InstallShield projects, ensure that all of you apply the SP1 patch at the same time.*

If you open an InstallShield 2012 Spring Advanced UI or Suite/Advanced UI project in InstallShield 2012 Spring SP1, InstallShield 2012 Spring SP1 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project before converting it.

Support for Adding Sideloading UWP App Packages (.appx) to Suite/Advanced UI Projects

InstallShield now lets you add sideloading UWP app packages (.appx) to a Suite/Advanced UI project. You can add this package type through the Packages view of the Suite/Advanced UI project just as you would add other types of packages to the project.

Sideloading an app is the process of installing an app without obtaining it through the Windows Store. This type of app is sometimes distributed to enterprise environments. Windows 8 and Windows Server 2012 include support for sideloading apps.

This support is available in the Premier edition of InstallShield.

Note that support for creating and building Suite/Advanced UI installations that include sideloading UWP app packages requires Windows Vista or later or Windows Server 2008 or later on the machine that has InstallShield or the Standalone Build.

For more information, see:

- Adding a Sideloading UWP App Package (.appx) to a Suite/Advanced UI Project
- Packages View

New UWP App Package Type of Condition Check in Suite/Advanced UI and Advanced UI Projects

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in a Suite/Advanced UI or Advanced UI project, you can select from a number of different types of checks that you want to be evaluated on target systems. Use the new UWP App Package type of condition check to check target systems for the presence of a particular UWP app package. The condition checks for a particular app name, and it can also check other information, such as the version.

For more information, see:

- Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects

- UWP App Package Condition Settings
- Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects

Support for Visual Studio 2012, .NET Framework 4.5, and Visual C++ 2012

InstallShield includes changes that offer support for the final released version of Visual Studio 2012, enabling development of installations and products within this version of the Visual Studio interface.

In addition, InstallShield includes two updated InstallShield prerequisites for the .NET Framework and two new InstallShield prerequisites for Visual C++:

- Microsoft .NET Framework 4.5 Full
- Microsoft .NET Framework 4.5 Web
- Microsoft Visual C++ 2012 Redistributable Package (x86)
- Microsoft Visual C++ 2012 Redistributable Package (x64)

You can add any of these prerequisites to Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects.

The Web prerequisite for the .NET Framework requires an Internet connection. This prerequisite downloads the required redistributable files if appropriate. The full prerequisite is a stand-alone installation that does not require an Internet connection.

Additional Changes

For a list of issues that are resolved in InstallShield 2012 Spring SP1, see the release notes. The release notes are available from the Help menu in InstallShield.

What's New in InstallShield 2012 Spring

New Features

InstallShield includes the following new features.

Ability to Target Windows 8 and Windows Server 2012 Systems

InstallShield enables you to specify that your installation requires Windows 8 or Windows Server 2012. It also lets you build feature and component conditions for these operating systems.

The InstallShield prerequisites that should be installable on Windows 8 and Windows Server 2012 have been updated so that they are installed on those systems if needed. Previously, the prerequisites were not run by default on those systems. This applies to the following InstallShield prerequisites:

- FSharp Redistributable Package 2.0
- Microsoft ReportViewer 2010
- Microsoft SQL CE 3.5 SP2
- Microsoft SQL Server 2005 Express SP3
- Microsoft SQL Server 2008 Express SP1

- Microsoft SQL Server 2008 Management Objects 10.00.2531
- Microsoft SQL Server 2008 Native Client 10.00.2531
- Microsoft SQL Server 2008 R2 Express RTM
- Microsoft SQL Server 2008 R2 Native Client 10.50.1600.1
- Microsoft SQL Server Native Client 9.00.4035
- Microsoft SQL Server System CLR Types 10.00.2531
- Microsoft Visual C++ 2005 SP1 Redistributable MFC Security Update KB2538242
- Microsoft Visual C++ 2005 SP1 Redistributable Package
- Microsoft Visual C++ 2008 SP1 Redistributable MFC Security Update KB2538243
- Microsoft Visual C++ 2008 SP1 Redistributable Package
- Microsoft Visual C++ 2010 Redistributable Package
- Microsoft Visual C++ 2010 RTM Redistributable MFC Security Update KB2467173
- Microsoft Visual C++ 2010 SP1 Redistributable Package
- Microsoft VSTO 2010 Runtime

Support for a New Contemporary, Customizable End-User Interface in InstallShield Professional Edition

The new Advanced UI project type lets you create a new end-user interface, with redesigned, contemporary wizard pages, for a Windows Installer package or an InstallScript package. This new project type is available in the Professional edition of InstallShield, and it is based on the technology that was previously introduced as the Suite project type (now known as the Suite/Advanced UI project type) in the Premier edition of InstallShield.

The new Advanced UI project type includes built-in wizard pages that you can include and customize in your Advanced UI installations. The wizard page editor in this project type lets you add, sequence, and remove pages as needed; it also lets you modify the layout of any page—adding, moving, and removing a variety of different kinds of controls. All of the new UI functionality that was previously available only through the Premier edition is now available through Advanced UI projects.

Like Suite/Advanced UI projects, an Advanced UI project uses the next-generation setup launcher (**Setup.exe**) for launching a package on target systems. Also like Suite/Advanced UI projects, an Advanced UI project includes flexible options for specifying the run-time source location of the package that you are including in the Advanced UI installation. The available location options are:

- On the Web, available for download by **Setup.exe** if needed
- Embedded in **Setup.exe** and extracted to the target system if needed
- Uncompressed and stored on the Advanced UI source media

Your end users can quickly download a small Advanced UI **Setup.exe** file, and the **Setup.exe** file can download and launch the Windows Installer–based or InstallScript package if needed.

Note that the Advanced UI project type includes support for including only one primary .msi package, one primary .msp package, or one primary InstallScript package. The Suite/Advanced UI project type that is available in the Premier edition of InstallShield includes support for packaging multiple primary installations (including .exe packages) as a single installation.

To learn more, see the following:

- Creating Advanced UI and Suite/Advanced UI Installations
- Advanced UI Projects vs. Suite/Advanced UI Projects
- Adding an .msi Package, an .msp Patch, or a Transaction to an Advanced UI or Suite/Advanced UI Project
- Working with the Wizard Interface

Microsoft Windows Azure SQL Database Support

InstallShield now includes support for running SQL scripts on Microsoft Windows Azure SQL database servers. In addition, InstallShield includes Microsoft Windows Azure in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports.

If your installation targets Windows Azure, the SQLBrowse run-time dialog that is displayed when end users choose to browse for a database catalog can now list catalogs on the specified SQL database server.

This support is available in the following project types: Basic MSI, DIM, InstallScript, and InstallScript MSI.

To learn more, see:

- Connecting to a Microsoft Windows Azure Database Server and Running SQL Scripts
- Configuring SQL Support
- SQL Scripts View

This support is available in the following project types: Basic MSI, DIM, InstallScript, and InstallScript MSI.

Beta Support for Microsoft Visual Studio 2012

InstallShield includes support for the beta of Visual Studio 2012. You can create InstallShield projects from within this version of Visual Studio.

Microsoft .NET Framework 4.5 Prerequisites

InstallShield includes two new .NET-related InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Microsoft .NET Framework 4.5 Full
- Microsoft .NET Framework 4.5 Web

These InstallShield prerequisites install the beta versions of the .NET Framework 4.5 on supported target systems.

The Web prerequisite requires an Internet connection. This prerequisite downloads the required redistributable files if appropriate. The full prerequisite is a stand-alone installation that does not require an Internet connection.

Microsoft SQL Server 2012 Support

InstallShield now includes support for running SQL scripts on SQL Server 2012 database servers. In addition, InstallShield includes SQL Server 2012 in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports.

If your installation targets SQL Server 2012, the SQLBrowse run-time dialog that is displayed when end users choose to browse for a database server can now list instances of SQL Server 2012, SQL Server 2012 Express, and SQL Server 2012 Express LocalDB. In addition, the SQLBrowse run-time dialog that is displayed when end users choose to browse for a database catalog can now list catalogs on the specified SQL Server 2012 database server.

To learn more, see:

- Requirements for Connecting to Instances of SQL Server Express LocalDB
- Configuring SQL Support
- SQL Scripts View

This support is available in the following project types: Basic MSI, DIM, InstallScript, and InstallScript MSI.

Microsoft SQL Server 2012 Prerequisites

InstallShield includes several new SQL Server 2012–related InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Microsoft SQL Server 2012 Express
- Microsoft SQL Server 2012 Express LocalDB
- Microsoft SQL Server 2012 Native Client

InstallShield also includes InstallShield prerequisites that install Microsoft .NET Framework 3.5 SP1 Update KB956250, which is a dependency of Microsoft SQL Server 2012 Express.

These InstallShield prerequisites install the technology on supported target systems.

New InstallShield Prerequisites for App-V 4.6 SP1, SQL Server Compact 4.0, and JRE SE 1.7

InstallShield includes new InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Java Runtime Environment Second Edition (JRE SE) 1.7
- Microsoft App-V 4.6 SP1 Desktop Client (The redistributable files for these InstallShield prerequisites—available in 32-bit and 64-bit versions—are not available for download from within InstallShield, since you must obtain them from Microsoft. Once you obtain them from Microsoft, place them in the location that is displayed when you are editing these prerequisites in the InstallShield Prerequisite Editor.)
- SQL Server Compact 4.0

These InstallShield prerequisites install the technology on supported target systems.

Support for the Configuring System Center 2012 Configuration Manager Application Model Data

Accurate identification of deployment metadata is necessary for migrating applications into the System Center 2012 Configuration Manager application model. InstallShield includes several new settings in the General Information view that let you specify some of the application model metadata for an application through a software identification tag. When AdminStudio users import a package into the AdminStudio Application Catalog, AdminStudio Application Manager mines package elements for deployment data such as detection methods, dependencies, requirements, as well as information in the software identification tag. AdminStudio makes this information available to users for review and tests before publishing to Microsoft System Center 2012 Configuration Manager.

This feature is available in Basic MSI projects.

To learn more, see Including Microsoft System Center Configuration Manager Application Model Data About Your Product.

Support for PowerShell Custom Actions

Windows PowerShell is a .NET Framework–based command-line shell and script language that enables system administrators to automate system configuration tasks. InstallShield now has support for custom actions that run PowerShell scripts. You may want to add this type of custom action to a project to perform system configuration tasks at installation run time.

Note that in order for an installation to run a PowerShell custom action, Windows PowerShell must be installed on target systems. InstallShield includes a new predefined PowerShell system search that checks for the presence of PowerShell on target systems. You can include this system search in your project and configure your PowerShell custom action to run only if the system search determines that PowerShell is installed.

The PowerShell execution policy, which determines whether PowerShell scripts can be run on a target system, is set to restricted by default, which does not permit PowerShell scripts to be run. If you want your installation to override the target system's execution policy with an appropriate one for your installation's PowerShell custom actions, you can use the new Windows Installer property **IS_PS_EXECUTIONPOLICY** to indicate the appropriate execution policy.

This feature is available in the following project types: Basic MSI and InstallScript MSI.

To learn more, see Calling a PowerShell Custom Action.

Automatic Update Check and Download Support for Suite/Advanced UI and Advanced UI Installations

Suite/Advanced UI and Advanced UI installations now have the ability to automatically check for an updated Suite/Advanced UI or Advanced UI **Setup.exe** file that you host on your Web site, and download and launch it if it is available. The updated Suite/Advanced UI or Advanced UI **Setup.exe** file can be used to deploy upgrades and patches for your latest Suite/Advanced UI and Advanced UI packages.

The Setup.exe tab in the Releases view of Suite/Advanced UI and Advanced UI projects has a new Update URL setting that lets you specify the absolute path (including the file name) for an updated Suite/Advanced UI or Advanced UI setup launcher. If the base Suite/Advanced UI or Advanced UI setup launcher is running a non-maintenance operation, the Suite/Advanced UI or Advanced UI setup launcher checks the update URL for a download. If a download is available, the base Suite/Advanced UI or Advanced UI setup launcher downloads it and then verifies its digital signature. If the digital signature in the update setup launcher matches that in the base setup launcher, the update setup launcher runs automatically. If the digital signature does not match, or if the base setup launcher is not digitally signed, a security warning is displayed, allowing the end user to choose whether to proceed with the update setup launcher.

To learn more, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.

Ability to Detect Whether a Specific Version of a Suite/Advanced UI or Advanced UI Installation Is Already Installed

Suite/Advanced UI and Advanced UI projects now include support for determining whether a particular version of a Suite/Advanced UI or Advanced UI installation is already installed on target systems. This type of condition check is called a Suite Installed condition.

InstallShield now includes two Suite Installed conditions in each Suite/Advanced UI and Advanced UI project by default:

- A new Suite Installed exit condition prevents end users from being able to install the current version of the Suite/Advanced UI or Advanced UI installation over a future newer version of the same Suite/Advanced UI or Advanced UI installation.
- A new Suite Installed mode condition may prevent the installation from running in first-time installation mode if the same version of the Suite/Advanced UI or Advanced UI installation is already installed.

These new default conditions are available in all new Suite/Advanced UI and Advanced UI projects. If you upgrade an InstallShield 2012 Suite project to InstallShield 2016, InstallShield automatically adds these default conditions to the project.

You can edit the default Suite Installed conditions if necessary. You can also add your own Suite Installed conditions to a project as needed.

For more information, see:

- Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed
- Triggering Install Mode or Maintenance Mode of an Advanced UI or Suite/Advanced UI Installation
- Suite Installed Condition Settings
- Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects

Ability to Configure Network Sharing of Folders in an Installation

InstallShield now lets you create, modify, and delete network shares to enable file sharing over networks. When you are configuring a folder in your project, you can indicate that you want to enable network sharing, which is disabled by default. You can also configure other options such as the name of the share, as well as the maximum number of simultaneous users who can access the share.

To enable sharing, use the new Sharing tab on the Properties dialog box, which is displayed when you right-click a folder in the Files and Folders view and then click Properties.

This feature is available in the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, and MSM Database.

For more information, see Folder Properties Dialog Box.

New Built-In Custom Action that Terminates Specific Processes

InstallShield includes support for a new kill-process type of custom action. If you add this type of custom action to your project, you can specify the name or process identifier (PID) of one or more processes that you want to be terminated at run time, and you can schedule the custom action for immediate or deferred mode.

The procedure for creating this type of custom action involves adding and configuring the custom action in the Custom Actions and Sequences view of your project, and using the Property Manager view to define a property with the names or PIDs of the appropriate processes.

This feature is available in the following project types: Basic MSI and InstallScript MSI.

For detailed instructions, see Calling a Kill-Process Custom Action.

Ability to Create Predetermined User Accounts and Groups at Run Time

InstallShield now has built-in support for creating multiple user accounts and corresponding groups at run time without using logon dialogs. To configure the accounts and groups, define values for the properties **ISNetApiLogonUsername**, **ISNetApiLogonGroup**, and **ISNetApiLogonPassword** in your project with the user names, groups, and passwords, respectively. Separate multiple names, groups, or passwords with a tilde enclosed by square brackets: [~].

This feature is available in the following project types: Basic MSI and InstallScript MSI.

To learn more, see Creating Predetermined User Accounts at Run Time.

Ability to Include InstallShield Prerequisites as Packages in Suite/Advanced UI and Advanced UI Projects

InstallShield now lets you import InstallShield prerequisites as .msi and .exe packages into Suite/Advanced UI and Advanced UI projects. You can import the InstallShield prerequisites that are included with InstallShield, as well as any custom InstallShield prerequisites that you have created. When you right-click the Packages explorer in the Packages view and then click the new Import Prerequisite (.prq) command, InstallShield adds to your project an .msi package or an .exe package, depending on the type of file that is configured to run for the prerequisite. InstallShield also automatically configures default values for each of the prerequisite package's settings, based on the settings that are configured in the .prq file. You can change these settings as needed, just as you can change the settings for packages in your Suite/Advanced UI or Advanced UI project.

If an InstallShield prerequisite has dependencies (that is, if one or more other .prq files are specified as dependencies in the InstallShield prerequisite that you are adding to the Suite/Advanced UI or Advanced UI project), InstallShield automatically adds the dependency prerequisites as separate packages in the Packages explorer.

Previously it was necessary to add the .msi and .exe installations as packages in a Suite project and then manually configure all of the conditions and settings for each of those packages.

For more information, see Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project.

Ability to Include InstallScript Installations as Packages in Suite/Advanced UI and Advanced UI Projects; New Suite/Advanced UI– and Advanced UI–Specific InstallScript Events and Functions

InstallShield now lets you add InstallScript installations as packages in Suite/Advanced UI and Advanced UI projects. When a Suite/Advanced UI or Advanced UI installation launches an InstallScript package, the Suite/Advanced UI or Advanced UI installation displays its own user interface (UI) while automatically suppressing the UI of the InstallScript package. This enables you to provide a contemporary UI experience for the installation. The Suite/Advanced UI or Advanced UI installation also displays progress information for the InstallScript package.

To make these changes possible, Suite/Advanced UI and Advanced UI installations use several new Suite/Advanced UI– and Advanced UI–specific InstallScript events and functions by default, and ignores some of the standard InstallScript events and functions.

New InstallScript Package Support in Suite/Advanced UI and Advanced UI Projects

InstallShield lets you add an InstallScript package to a Suite/Advanced UI or Advanced UI project if the InstallScript package meets the following requirements:

- The InstallScript package is uncompressed.
- InstallShield 2012 Spring or later is used to build the InstallScript package and the Suite/Advanced UI or Advanced UI installation.

- The InstallScript package uses an event-based script; it should not use the program...endprogram style script.

For more information, see Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project.

New Suite/Advanced UI– and Advanced UI–Specific InstallScript Events and Functions

In a standard InstallScript installation that is launched via the InstallScript **Setup.exe** file (that is, not launched from a Suite/Advanced UI or Advanced UI installation), most events are called directly from the OnShowUI event. In a Suite/Advanced UI or Advanced UI installation that launches an InstallScript package, OnShowUI is replaced with OnSuiteShowUI. Depending on the installation state (first-time installation, maintenance, or update), OnSuiteShowUI ignores the UI events such as OnFirstUIBefore and OnFirstUIAfter and instead calls the following events:

- **First-time installation**—OnSuiteInstallBefore, OnSuiteInstallAfter
- **Maintenance**—OnSuiteMaintBefore, OnSuiteMaintAfter
- **Update**—OnSuiteUpdateBefore, OnSuiteUpdateAfter

The InstallScript language includes some new Suite/Advanced UI– and Advanced UI–specific functions that enable interaction between an InstallScript package and the Suite/Advanced UI installation or the Advanced UI installation that is running it. For example, InstallScript includes new functions that let you log InstallScript package information to Suite/Advanced UI and Advanced UI debug logs, set and retrieve Suite/Advanced UI and Advanced UI properties, and pass data from Suite/Advanced UI and Advanced UI installations to the InstallScript package.

To determine whether the InstallScript installation is running as an InstallScript package in a Suite/Advanced UI or Advanced UI installation, use the new SUITE_HOSTED variable in your InstallScript code.

New InstallScript Package Type of Condition Check in Suite/Advanced UI and Advanced UI Projects

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in a Suite/Advanced UI or Advanced UI project, you can select from a number of different types of checks that you want to be evaluated on target systems. Use the new InstallScript package type of condition check to check target systems for the presence of a product that was installed by a particular InstallScript installation. The condition checks for a particular product code, and it can also check other information, such as the product version.

To learn more, see the following:

- Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects
- InstallScript Package Condition Settings
- Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects

Dynamic File Link Support for Package Files in Suite/Advanced UI and Advanced UI Projects

When you are adding or configuring an .msi, .msp, or .exe package in an Advanced UI or Suite/Advanced UI project, you can indicate whether the package requires additional files that are located near the package file. For example, if a package that you are adding is an uncompressed .msi package, you may need to include other files—such as .cab files and uncompressed data files in nearby subfolders—along with the package file.

InstallShield now lets you use dynamic links for the additional package files. Dynamic links are useful if the list of additional files that the package requires is likely to change between builds. InstallShield scans the source folder before every build and automatically incorporates any new or changed package files in your release.

InstallShield also lets you define filters that control which additional files InstallShield should include in the dynamic link at build time, and which ones InstallShield should exclude. You can change the order in which InstallShield evaluates any filter criteria that you have defined for a dynamic link. Each time that you build your Advanced UI or Suite/Advanced UI installation, InstallShield includes the appropriate additional files based on the dynamic link's filters.

Previously, only static links could be used for additional files. If the list of additional files changed between builds, it was necessary to manually update the list of package files.

For more information, see:

- Static vs. Dynamic Files for Packages in an Advanced UI or Suite/Advanced UI Project
- Creating a Dynamic Link in an Advanced UI or Suite/Advanced UI Project
- Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project

Ability to Give Enhanced Feedback When Validating End-User Input During a Suite/Advanced UI or Advanced UI Installation

Suite/Advanced UI and Advanced UI installations now have support for providing enhanced feedback when validating end-user input at run time. Various interface controls in the Wizard Interface view of a Suite/Advanced UI project and an Advanced UI project have three new subsettings under the existing Text Style setting: Default, Valid, and Invalid. You can configure these subsettings to select different text styles that you want the Suite/Advanced UI or Advanced UI installation to use under different circumstances.

Support for Creating Package Log Files When Launching a Suite/Advanced UI or Advanced UI Installation from the Command Line

When you are configuring the settings for a package in a Suite/Advanced UI or Advanced UI project, you can use the new Enable Logging Support setting to specify whether you want the package to generate a log file if the Suite/Advanced UI or Advanced UI installation is launched from the command line with the new `/l*og` command-line parameter. Depending on the type of package (.msi package, .msp package, or some other type of package), you can also configure one or two other settings to specify information such as which log options you want the Suite/Advanced UI or Advanced UI installation to pass to the package when the log file is being created.

The new `/l*og` command-line parameter for the Suite/Advanced UI or Advanced UI **Setup.exe** file lets you specify the path to the directory that contains the package log files. If a path is not specified with the `/l*og` parameter, the Suite/Advanced UI or Advanced UI installation creates the package log files in the %TEMP% directory.

The new property **ISLogDir** in Suite/Advanced UI and Advanced UI installations stores the path to that directory that contains the package log files.

Previously, the only way to enable logging for a Windows Installer–based package in a Suite installation was to use the logging system policy or the **MsiLogging** property.

For more information, see:

- Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation
- Advanced UI and Suite/Advanced UI Setup.exe Command-Line Parameters
- Advanced UI and Suite/Advanced UI Property Reference

Support for 64-Bit Components in InstallScript Installations

InstallScript projects now have support for installing files to WINSYSDIR64 (the InstallScript variable that maps to the 64-bit System32 folder), and for installing registry data to the 64-bit registry locations, without requiring you to modify your InstallScript code. If you have files or registry data that need to be installed to these 64-bit locations, you can add the files and registry data to a component, and select Yes for that component's new 64-Bit Component setting. At run time, the installation automatically disables file system redirection for the component's System32 files, and it prevents redirection for the component's 64-bit registry data.

Previously, to install files to WINSYSDIR64, it was necessary to override the Installing and Installed events for features that contained components that installed to that location. In the Installing event, it was necessary to use the WOW64FSREDIRECTION constant with the **Disable** function to disable file system redirection; in the Installed event, it was necessary to use WOW64FSREDIRECTION with the **Enable** function to re-enable file system redirection for other parts of the installation. The same sort of disabling and enabling was necessary for the UnInstalling and UnInstalled events to ensure that those files were removed correctly during uninstallation.

If file system redirection is not disabled when an InstallScript installation installs to WINSYSDIR64, 64-bit Windows automatically redirects the file transfers to the 32-bit System32 folder (SysWOW64).

Also previously, to install registry data to a 64-bit area of the registry, it was necessary to use the InstallScript registry functions to create registry data with REGDB_OPTION_WOW64_64KEY set in REGDB_OPTIONS. Then it was necessary to use REGDB_OPTION_USE_DEFAULT_OPTIONS with REGDB_OPTIONS to re-enable registry redirection for other parts of the installation.

If registry redirection is not disabled when an InstallScript installation installs to a 64-bit registry location (HKEY_LOCAL_MACHINE\Software), 64-bit Windows automatically redirects the registry changes to the equivalent 32-bit registry location (HKEY_LOCAL_MACHINE\Software\Wow6432Node).

The InstallScript log files (.ilg) that InstallScript installations create at run time when installing a product now use a new OPTYPE_FILE64 type to identify files that are installed when file system redirection is disabled. The InstallScript log files use the existing OPTYPE_REGISTRY64 type to identify registry data that are installed when registry redirection is disabled. You can see these OPTYPE_FILE64 and OPTYPE_REGISTRY64 types when viewing an .ilg file in the InstallShield Cabinet and Log File Viewer.

For more information, see the following:

- Targeting 64-Bit Operating Systems with InstallScript Installations
- Component Settings
- WINSYSDIR64
- REGDB_OPTIONS
- InstallShield Cabinet and Log File Viewer

New Locale Type of Condition Check in Suite/Advanced UI and Advanced UI Projects

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in a Suite/Advanced UI or Advanced UI project, you can select from a number of different types of checks that you want to be evaluated on target systems. Use the new locale type of condition check to check for matching one or more locale-related settings on target systems.

To learn more, see:

- Using Locale Conditions in Advanced UI and Suite/Advanced UI Projects

- Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects
- Locale Condition Settings
- Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects

New Wizard Interface Toolbar for Editing the Layout in Suite/Advanced UI and Advanced UI Projects, with Support for Switching Languages in Suite/Advanced UI Projects

If you select a wizard page or secondary window in the Wizard Interface view of a Suite/Advanced UI project, the toolbar that InstallShield shows directly above the wizard interface preview pane includes several different buttons and other controls that let you modify the layout of the selected page or window. InstallShield also displays this toolbar in the Wizard Interface view if you select one or more controls on a wizard page or a secondary window.

The new toolbar has buttons that let you add labels, text boxes, check boxes, and other controls to the installation's user interface. The toolbar also has buttons that let you easily align selected controls, resize them, and position them in relation to each other. In Suite/Advanced UI projects, the Default Languages list in the new toolbar enables you to switch the strings that InstallShield displays on the wizard pages and secondary windows in this view to those in a different language in your project.

For more information, see:

- Adding a Control to a Wizard Page or Secondary Window
- Wizard Interface View Toolbar

Support for Adding Languages to Suite/Advanced UI Projects

The InstallShield Premier edition includes default run-time strings in 35 supported languages. When you add a supported language to a Suite/Advanced UI project, that language is made available in various language-related settings throughout InstallShield. In addition, InstallShield adds translated string entries for that language to your project. The string entries are for the default wizard pages, messages, and other end-user interface elements.

InstallShield now lets you add unsupported languages, beyond the built-in 35 languages, to Suite/Advanced UI projects through the New Language Wizard. An unsupported language is one in which none of the default run-time strings are translated. When you add an unsupported language to a Suite/Advanced UI project, that language is made available in various language-related settings throughout the project. In addition, InstallShield uses the strings from your project's default language as placeholders for the strings in that newly added unsupported language; you can use the String Editor view to provide translated strings for the unsupported languages.

To launch the New Language Wizard in a Suite/Advanced UI project, on the Tools menu, click Add New Language.

To learn more, see the following:

- New Language Wizard
- Run-Time Language Support in InstallShield

Ability to Create and Configure Scheduled Tasks on Target Systems at Run Time

InstallShield has a new Scheduled Tasks view that lets you configure one or more tasks that you want to be created through the Windows task scheduler at run time on target systems. The view lets you specify information such as the file that you want to be launched for a task, as well as the start date and time. The file that you want to be launched can be part of your installation, or it can be a file that is already present on target systems.

This feature is available in the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, Transform.

For more information, see:

- Scheduling Tasks
- Scheduled Tasks View

New FlexNet Connect 13.03 Redistributables Available

InstallShield includes support for FlexNet Connect 13.03 in Basic MSI and InstallScript MSI projects. Use the Update Notifications view in InstallShield to include one of the two FlexNet Connect 13.03 merge modules—one has the Common Software Manager, and the other does not.

Enhancements

InstallShield includes the following enhancements.

Enhancements to the InstallScript Language for Operating Systems

The following structure members and predefined constants were added to the InstallScript language:

- **SYSINFO.WINNT.bWin8**—This is a new SYSINFO structure member. If the operating system is Windows 8 or Windows Server 2012, this value is TRUE.
- **ISOSL_WIN8**—This is a new predefined constant that is available for use with the **FeatureFilterOS** function and the SYSINFO structure variable. It indicates that the target system is running Windows 8 or Windows Server 2012.

For more information, see FeatureFilterOS function and the SYSINFO structure variable.

Automation Interface Enhancement: OSFilter Property Value for Windows 8 and Windows Server 2012

The following constants are now available for use with the OSFilter member of the ISWiComponent and ISWiRelease objects in the automation interface:

eosWin8 = &H4000000 (67108864)—These are for Windows 8 and Windows Server 2012.

In addition, the value for the eosAll constant is now &7D100D0 (131137744); previously, it was &3D100D0 (64028880).

The OSFilter member applies to the ISWiComponent object in InstallScript, InstallScript MSI, and InstallScript Object projects. The OSFilter member applies to the ISWiRelease object in InstallScript and InstallScript Object projects.

For more information, see the following:

- ISWiComponent Object
- ISWiRelease Object

Ability to Easily Move Conditions in Suite/Advanced UI and Advanced UI Projects

When you have more than one conditional statement for an exit, detection, eligibility, or feature condition in a Suite/Advanced UI or Advanced UI project, you now can move the conditional statements to reorder conditions or change the hierarchy of a condition tree. For example, if you have a platform conditional statement in a None condition group, and the None condition group is in an All condition group, you can move the platform conditional statement left, so that it is only part of the All condition group.

To move a conditional statement or group, click the new Move Condition button in the setting of the item that you want to move, and then click the appropriate option (Move Up, Move Down, Move Left, or Move Right).

Previously, it was necessary to manually create the new conditional statement in the new location and delete the one in the old location. Or, as an alternative, you could edit the .issuite file in a text editor and change the order of the conditional statements.

For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.

New Extension Condition Support and Enhanced Condition Settings in Suite/Advanced UI and Advanced UI Projects

In Suite/Advanced UI and Advanced UI projects, new extension condition functionality is available. This type of condition lets you browse to a C/C++ DLL that you have created to check for your own custom conditions on target systems.

In addition, many of the condition-related settings that are available in Suite/Advanced UI and Advanced UI projects have been enhanced to make it easier to build conditional statements. For example, instead of manually trying to enter a product code, upgrade code, patch code, or other various types of data in any one of several of the condition settings, you can now click a new ellipsis button (...) in the setting; when you do this, InstallShield displays a browse dialog box that lets you browse to and select the appropriate package. Once you have selected a package, InstallShield enters the appropriate information from that package in the setting of the Suite/Advanced UI or Advanced UI project.

When you are configuring an eligible package condition, you can now select from a list of packages in your Suite/Advanced UI or Advanced UI project instead of having to manually enter the package GUID of the appropriate package.

For more information, see:

- Creating a Custom Condition for an Advanced UI or Suite/Advanced UI Installation
- Creating an Extension Condition DLL for an Advanced UI or Suite/Advanced UI Project
- Adding an Extension Condition DLL to an Advanced UI or Suite/Advanced UI Project
- Extension Condition Settings

Enhancements for Configuring Validation and Actions for a Control on a Wizard Page or Window in Suite/Advanced UI and Advanced UI Projects

In Suite/Advanced UI and Advanced UI projects, the Validation and Action settings for various controls on the wizard interface have been enhanced to make it easier to define validation and trigger various actions. These settings now contain drop-down lists of sample statements that you can enter in these settings; these settings are also still text boxes that let you enter statements manually. For example, the Validation setting lets you specify a format that end users must match when they enter a serial number in a control. As an alternative to manually entering the entire validation statement, you can now select the mask type of validation in the Validation setting and then override the default format in the setting.

Similarly, the Action setting lets you define—for example—a print action for a button control; when an end user clicks the Print button, the Print dialog box opens, enabling the end user to print the license agreement. As an alternative to manually entering the action statement, you can now select the print action in the Action setting and then override the file name with the name of the file that you want to be printed.

The drop-down lists in the Validation and Action settings also now include C/C++ DLL files that you have created and added to your project through the Support Files view. This functionality lets you trigger your own validation or your own action for various controls on the wizard interface.

To learn more, see:

- Configuring Validation for a Control on a Wizard Page or Window
- Configuring an Action for an Element in the Wizard Interface

Ability to Use Custom Folder Names for Packages in Suite/Advanced UI and Advanced UI Releases

When you build an Suite/Advanced UI or Advanced UI installation, InstallShield creates a folder for each package that is included in the installation; these folders are created in the same folder that contains the Suite/Advanced UI or Advanced UI setup launcher. By default, the name of each folder is a GUID that InstallShield generates.

The Packages view in InstallShield now lets you override the GUID name with a user-friendly name for each package folder. To enter a custom folder name in this view, find a Package Files folder under the package whose folder name you want to customize. Right-click that folder, click Rename, and enter a new name. If you customize more than one folder name, ensure that each folder name is different.

Previously, InstallShield used a GUID for the name of each folder and did not have support for customizing the names.

For instructions, see Using Custom Folder Names for Packages in Advanced UI and Suite/Advanced UI Installations.

New Formatted Support for Properties Whose Values Reference Other Properties in Suite/Advanced UI and Advanced UI Projects

Each property that is defined in the Property Manager view in a Suite/Advanced UI or Advanced UI project has a new Formatted check box. This new check box lets you indicate whether you want the properties that are referenced in a property's value to be resolved and replaced by their property values at run time.

To replace properties that are enclosed within square brackets (such as **[PropertyName]**) at run time, select this check box. To leave square brackets and the content within them as is, clear this check box.

For more information, see Property Manager View.

Improved Timing for UAC Prompts of Downloaded Packages that Require Elevation for Suite/Advanced UI and Advanced UI Installations that Have an Invoker Manifest

If you build an Advanced UI or Suite/Advanced UI release with an Invoker manifest, and if Yes is selected in the Require Elevated Privileges setting for any of the installation's packages that need to be downloaded and launched on the target system, the installation triggers the UAC prompt soon after end users click the Install button—before the download occurs.

Previously, the installation triggered the UAC prompt after the download occurred. Thus, if package staging was slow (for example, if the package download took a long time), there could be a big gap between the moment that an end user clicked the Install button and the moment that Windows displayed the UAC prompt for elevation for the package. If an end user did not provide consent or credentials quickly enough, the UAC prompt timed out, and the installation failed.

In some previous cases, using an Administrator manifest was a possible workaround, since the UAC prompt was displayed soon after end users clicked the Install button. However, with this workaround, the entire installation had elevated privileges.

Support for Specifying the Alignment for Text in the Wizard Interface of Suite/Advanced UI and Advanced UI Installations

InstallShield has a number of built-in text styles that define text attributes such as color, size, and font name for the text on the wizard interface of Suite/Advanced UI and Advanced UI projects. You can edit any of the settings for these built-in styles or define your own styles, through the Wizard Interface view in your Advanced UI or Suite/Advanced UI project.

Each built-in or custom text style includes a new Text Alignment setting that lets you select the type of alignment that you want to use for the text in controls that use that particular style.

To learn more, see Using Styles, Brushes, and Control Themes to Customize the Wizard Interface.

Enhanced Combo Box Controls for the Wizard Interface of Suite/Advanced UI and Advanced UI Installations

InstallShield lets you add a combo box control to a wizard page or a secondary window in Suite/Advanced UI and Advanced UI projects. This type of control is a combination of two controls by default:

- A box that contains a drop-down list of predefined values
- A text box that lets end users enter a custom value

Previously, if you added a new combo box control, the control contained a drop-down list, but it was not also a text box; that is, end users could not enter a custom value.

To change this control to a drop-down list without the text box (that is, if end users should be able to select a predefined value but not enter a custom value), set the CBS_DROPDOWNLIST style for this control to True.

Enhancements and Changes to the Aero Format for the Suite/Advanced UI and Advanced UI Wizard Interface

Some enhancements and changes have been made to Aero-formatted wizard pages in Suite/Advanced UI and Advanced UI installations:

- The header and navigation areas of wizard pages are now displayed with the Aero glass effect, or translucency. Previously, only the caption bar of wizard pages were displayed with the Aero glass effect.
- The Aero-formatted wizard pages now use the same layout as Wizard 97–formatted wizard pages. That is, the caption bar on Aero-formatted wizard pages is no longer extra tall; it is the same height as the caption bar on Wizard 97–formatted wizard pages. In addition, the Back button on Aero-formatted wizard pages is now displayed in the navigation area, which is consistent with the placement on Wizard 97–formatted wizard pages. Previously, the top-left corner of the caption bar in Aero-formatted wizard pages contained the Back button.

Ability to Remove a String Value and Its Identifier from a Setting

When you are entering the value of a setting in one of the views in InstallShield and that value is a text string that can be presented to end users, InstallShield automatically uses a string identifier for that setting. InstallShield places the string identifier in curly brackets before the string value. These types of settings now contain a new **Delete this string reference** button.

If you want to remove the string identifier and its value from a setting, you can now click this new button. The button lets you clear the entry in the setting. Note that if you want to delete the string identifier and its value from your project, you must use the String Editor view.

What's New in InstallShield 2012 SP1

Enhancements

InstallShield includes the following enhancement.

Support for Digitally Signing Software Identification Tags

If you configure your project to include a software identification tag and you also configure the release in the Releases view to use a .pfx file to digitally sign your release, InstallShield digitally signs the tag at build time. Note that the .NET Framework 2.0 or later must be installed on your build machine in order to sign a tag file.

For more information, see Including a Software Identification Tag for Your Product.

What's New in InstallShield 2012

New Features

InstallShield includes the following new features.

Ability to Create Suite Installations that Run Multiple Packages; New Contemporary, Customizable End-User Interface; Ability to Build Hybrid 32-Bit/64-Bit Installations

The Premier edition of InstallShield now lets you build a Suite installation that uses the next-generation setup launcher (**Setup.exe**) to conditionally run multiple installations and apply Windows Installer patches (.msp) as needed on target systems. This support is available through a new Suite project type. Suite installations can be run on systems with Windows XP and later and with Windows Server 2003 and later.

Following are some highlights of this functionality.

Ability to Package Multiple Installations as a Single Installation

The new Suite project type contains a Packages view that lets you specify one or more of the following types of packages:

- Executable files (.exe), including Windows Installer–based and non-Windows Installer–based installations that you want to be run on target systems
- Windows Installer packages (.msi) that you want to be run on target systems
- Windows Installer patches (.msp) that you want to be applied on target systems

The Packages view also lets you include multiple .msi and .msp packages that you want to be run using transaction processing, a feature of Windows Installer 4.5 and later. The packages are chained together and processed as a single transaction. Each Suite installation can have multiple separate transactions. If one or more of the packages in a transaction cannot be installed successfully or if the end user cancels the installation, Windows Installer initiates rollback for all of the chained packages within the current transaction to restore the system to its earlier state.

The Suite installation launches the appropriate packages at run time based on conditions that you have defined and the order in which you listed the packages in the Packages view.

Contemporary, Customizable User Interface for the Installation; New Editor for Customizing Suite Setup.exe Wizard Pages

The Suite project type in InstallShield includes an entirely new end-user interface, with redesigned, contemporary built-in wizard pages that you can include and customize in your Suite installations. The new wizard page editor in this project type lets you add, sequence, and remove pages as needed; it also lets you modify the layout of any page—adding, moving, and removing a variety of different kinds of controls.

Support for Combining 64-Bit and 32-Bit Windows Installer Packages Into a Single Installation

As more and more users move to 64-bit versions of Windows, you may need to now, or in the near future, deliver to customers a single installation that installs to 32-bit locations on 32-bit systems and to 64-bit locations on 64-bit systems. The Suite project type lets you include both 32-bit packages and 64-bit packages in one Suite installation, and run only the appropriate packages on each target system. Previously, other alternatives required delivering two separate installations (one for 32-bit systems and one for 64-bit systems) or creating a custom launcher, a bootstrap application, or an InstallScript installation.

Support for Displaying a Single Progress Bar that Shows the Overall Status of the Entire Suite of Packages

The progress bar on the progress wizard page in a Suite installation shows the status of the entire suite of packages. This integrated progress bar presents end users with a clear visual indication of how far along the Suite installation is overall. To ensure that end users see only the integrated progress bar, you must include only .exe installations for which you have specified command-line parameters that hide the user interface (that is, run silently), .msi packages, and .msp patches.

Optional Add or Remove Programs Entry for the Suite Installation

Suite projects let you specify whether you want to have an entry in Add or Remove Programs for your Suite installation. This entry lets end users perform maintenance for your Suite, modifying or removing if needed. The General Information view in a Suite project has a Show Add or Remove Programs Entry setting that lets you indicate the appropriate behavior.

If you want to show only a single entry for the entire Suite, ensure that you hide the entries from the packages that you include in the Suite project.

Support for Running Suite Installations Without a User Interface

End users can run your Suite installations with a user interface or silently, without a user interface. Silent installations run without user intervention; end users can avoid monitoring the installation and providing input through run-time wizard pages.

Default Setup.exe User Interface Strings Included for All 35 Supported Languages; Ability to Edit the Suite Run-Time Strings

Translations of all of the default strings that are displayed in the built-in wizard pages of Suite projects are available in all 35 of the run-time languages that InstallShield supports. All of these Suite strings are displayed in the String Editor view of a Suite project. This String Editor view offers the same robust support that other types of projects offer, giving you complete and centralized control over the text strings that are displayed at run time during the Suite installation process.

Small Base Setup.exe File with the Ability to Download Only the Required Packages As Needed

Suite projects include flexible options for specifying the run-time source location of each package in the Suite installation. When you define the packages that you are including in a Suite project, you can specify the location of each individual package. The available options are:

- On the Web, available for download by **Setup.exe** if needed
- Embedded in **Setup.exe** and extracted to the target system if needed
- Uncompressed and stored on the Suite source media

The base **Setup.exe** file that is used for Suite installations is much smaller than the base **Setup.exe** files that are used for Basic MSI, InstallScript MSI, and InstallScript installations. Thus, your end users can quickly download a small Suite **Setup.exe** file, and the **Setup.exe** file can download and launch one or more required packages as needed.

For more information, see:

- Creating Advanced UI and Suite/Advanced UI Installations
- Adding an .msi Package, an .msp Patch, or a Transaction to an Advanced UI or Suite/Advanced UI Project
- Adding an Executable Package (.exe) to a Suite/Advanced UI Project
- Working with the Wizard Interface
- Advanced UI and Suite/Advanced UI Setup.exe Command-Line Parameters
- Advanced UI and Suite/Advanced UI Property Reference
- Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects

Redesigned, Expanded Support for Modularizing Installation Projects to Enable Reuse and Distribute Development Work

InstallShield includes a new project type called DIM, which is known as a *developer installation manifest*. A DIM project is a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete portion of a product installation. Some benefits of using DIMs are as follows:

- DIMs include support for virtually the same functionality that is available in Basic MSI projects. This gives authors of DIMs all of the flexibility that they need to develop their portions of an installation.
- Release engineers can reuse DIMs in multiple Basic MSI projects, enabling efficiency.
- Working with DIMs enables multiple team members to contribute to the development of the installation simultaneously. Each software developer or other team member can work on a separate DIM that the release engineer can reference in one or more Basic MSI projects.

Once you have created a DIM, you can add it to a Basic MSI project in one of two ways:

- **By reference**—You can add a reference for a DIM project to your Basic MSI project through the Setup Design view or the DIM References view. With this method, the DIM elements are merged into the Basic MSI project at build time. Each time that you build the Basic MSI installation, InstallShield references the latest version of the DIM project and includes it in the installation that it generates. This method is the more commonly performed method.

- **By import**—You can import a DIM project into your Basic MSI project by using the new Import DIM Wizard. This method is a one-time, irreversible import that merges the DIM data into the Basic MSI project at design time.

This redesigned, expanded DIM support replaces the previous support that required users to create DIMs in a separate tool called InstallShield Collaboration, and then import the DIM files into InstallShield Basic MSI projects. The new DIM support is more robust than the previous support. The new DIM project type lets you have virtually the same complete level of authoring flexibility that is available for Basic MSI projects. For example, the new DIM project type lets you have full control over component creation: you can add components to a new DIM project, set the key file of a component, and configure the component's settings. The new DIM project type also lets you configure IIS Web sites. InstallShield Collaboration did not have that sort of flexibility over component design, and it did not have any built-in support for configuring IIS Web sites.

The ability to create DIM projects is available in the Premier edition of InstallShield. This support is also available in the InstallShield Developer Installation Manifest Editor, a new collaboration add-on. The ability to add DIM files to Basic MSI projects is available in the Premier edition of InstallShield.

For more information, see the following:

- Modularizing Installation Projects to Distribute Development Work and Enable Reuse
- Including DIMs in a Project
- Determining the Appropriate Method for Incorporating DIMs in Installation Projects
- Using Path Variables in a DIM
- DIM References View
- Import DIM Wizard

New InstallShield Prerequisites for Internet Explorer 9, SQL Server 2008 R2 Native Client, Windows Identity Foundation, and Other Redistributables

InstallShield includes several new InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Internet Explorer 9.0
- Microsoft SQL Server 2008 R2 Native Client 10.50.1600.1
- Windows Identity Foundation
- Microsoft VSTO 2010 Runtime (x64)
- Microsoft Office 2010 PIA (This prerequisite installs the Microsoft Office 2010 Primary Interop Assemblies. To use this prerequisite, download the **PrimaryInteropAssembly.exe** file from Microsoft's Web site and run it to extract the .msi file.)

Support for 64-Bit Dependency Scanning

The Static Scanning Wizard and the Dynamic Scanning Wizard—dependency scanners in InstallShield—now include support for identifying 64-bit dependencies of the 64-bit files in your project. If you are using InstallShield on a 64-bit version of Windows Vista or later or a 64-bit version of Windows Server 2008 or later, the scanners can detect the 64-bit dependencies. The wizards let you specify whether you want to include each detected potential dependency in your project.

In addition, if you use InstallShield on a 64-bit version of Windows Vista or later or a 64-bit version of Windows Server 2008 or later, and you use either of the following built-in methods for detecting dependencies, InstallShield can scan for 64-bit dependencies of the 64-bit .NET assemblies in your project:

- The Static Scanning Wizard helps you identify a 64-bit .NET assembly's possible dependencies on demand. This wizard displays a list of the dependencies that it finds, and it lets you specify whether you want to include each one in your project.
- A component's .NET Scan at Build setting lets you specify whether you want InstallShield to identify a 64-bit .NET assembly's dependencies each time that you build your project. For InstallScript projects, the component's .NET Assembly setting must also be set to Local Assembly. If InstallShield detects any possible missing dependencies at build time, InstallShield incorporates them into the release that it generates.

InstallShield must be installed on a 64-bit operating system in order to scan 64-bit files for 64-bit dependencies. Note that if you use InstallShield on a 32-bit version of Windows, these built-in scans can check for only 32-bit dependencies of the 32-bit files in your project. If your project includes 64-bit files, you can manually add any dependencies to the project as needed.

To learn more, see:

- Developing and Building Installations on 32-Bit vs. 64-Bit Systems
- Scanning for 64-Bit Dependencies
- Identifying Application Dependencies
- Scanning 64-Bit .NET Assemblies for Dependencies
- Identifying Properties and Dependencies of .NET Assemblies

Support for Setting Permissions for Files, Folders, and Registry Keys in 64-Bit Locations

InstallShield has support for setting permissions for files, folders, and registry keys in 64-bit locations. The support varies, depending on which project type you are using.

Using the Custom InstallShield Handling Method in Windows Installer–Based Projects

If you are using the custom InstallShield handling method to set permissions for files, folders, and registry keys, you can now set permissions for these items when they are in 64-bit locations; this includes 64-bit locations in the registry, as well as the 64-bit System32 folder on 64-bit systems. The file, folder, or registry key must be included in a component that is marked as 64 bit (that is, Yes is selected for its 64-Bit Component setting). Previously, if the component was marked as 64 bit, permissions could not be set, and a run-time error was displayed.

This custom InstallShield handling support is available in the following project types: Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform. The Locked-Down Permission setting in the General Information view lets you specify which method you want to use for setting permissions: either the custom InstallShield handling method or the traditional Windows Installer handling method.

Using the InstallScript Function SetObjectPermissions in InstallScript-Based Projects and Windows Installer–Based Projects

If the REGDB_OPTION_WOW64_64KEY option is enabled and you use the InstallScript function **SetObjectPermissions** to set permissions for a 64-bit registry key, the function can set its permissions correctly. To force **SetObjectPermissions** to set permissions for a 64-bit registry key regardless of whether the

REGDB_OPTION_WOW64_64KEY option is enabled, you can use the new IS_PERMISSIONS_OPTION_64BIT_OBJECT constant; pass this constant in the nOptions parameter of **SetObjectPermissions**. Note that the IS_PERMISSIONS_OPTION_64BIT_OBJECT constant should not be passed on 32-bit target systems.

If file system redirection is disabled using the WOW64FSREDIRECTION constant when **SetObjectPermissions** is called to set permissions for a file or folder in the 64-bit System32 folder, the function can set the permissions correctly. If file system redirection is not disabled, the permissions cannot be set.

You can use the **SetObjectPermissions** function in InstallScript events in InstallScript and InstallScript MSI projects. You can also use this function in InstallScript custom actions in the following project types: InstallScript, Basic MSI, InstallScript MSI, and Merge Module.

For more information, see SetObjectPermissions.

Improvements for COM Extraction

InstallShield supports a new monitoring method for COM extraction. If you are using InstallShield on a Windows Vista or later system or a Windows Server 2008 or later system, this new method is used by default. The method uses a kernel driver to monitor the areas of the registry that are modified during dynamic COM extraction at build time and static COM extraction at design time. It combines the advantages that the earlier methods provided, allowing the DLL to read existing registries entries and preventing changes to the build machine.

If necessary, you can switch between the three different COM extraction methods by setting the value data of the UseAPIRegistryHooks registry value, which is in the registry key HKEY_LOCAL_MACHINE\SOFTWARE\InstallShield\RegSpy (on 32-bit machines) or HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\InstallShield\RegSpy (on 64-bit machines). Possible REG_DWORD value data are:

- **0**—Use API hooking to read existing registry entries for the DLL.
- **1**—Use registry redirection to prevent making changes to the registered DLLs on the build machine. If the value is not set, this is the default behavior on Windows XP and Windows Server 2003 systems.
- **2**—Use the new kernel mode monitoring, which combines the advantages of both of the other methods. If the value is not set, this is the default behavior on Windows Vista and later systems and on Windows Server 2008 and later systems.

This feature applies to the following project types: Basic MSI, DIM, InstallScript MSI, and Merge Module.

Predefined System Searches for Adobe Reader 10, Internet Explorer 9, and Microsoft Office

InstallShield has new predefined system searches:

- Adobe Reader 10
- Internet Explorer 9
- Microsoft Office 2010
- Microsoft Office 2007
- Microsoft Office 2003

If your installation requires one or more of these, you can use the System Search view or the Installation Requirements page in the Project Assistant to add these system searches to your project. When end users launch your installation, Windows Installer checks the target system to see if the requirements are met; if they are not met, the installation displays the error message that is defined for the system search.

This feature applies to Basic MSI and InstallScript MSI projects.

Support for Software Identification Tagging

ISO/IEC 19770-2 is an international standard for the creation of software identification tags. A software identification tag is an XML-based file that contains descriptive information about the software, such as the product name, product edition, product version, and publisher. Software asset management tools collect the data in the tags to provide accurate application identification for software that is installed in an enterprise.

Software identification tagging is evolving as an industry standard, enabling independent software vendors to create smarter applications that give their customers better information for software asset management and license optimization initiatives. Including the identification tag in your product's installation makes it possible for your customers to use tools that can monitor their internal usage of your product, allowing them to manage and optimize the number of licenses of your product that they obtain from you, and stay in compliance with your licensing policies.

InstallShield includes several new settings in the General Information view that let you specify information that is required to create an identification tag for your product. This view also contains a new Use Software Identification Tag setting that lets you specify whether you want InstallShield to create the tag at build time and include it in your installation; the default value of this setting is Yes.

Note that if Yes is selected for the Use Software Identification Tag setting but you have not entered values in one or more of the required identification settings (the Unique ID, Tag Creator, and Tag Creator ID settings in the General Information view), build warning -7235 occurs, once for each of the settings that are blank. This build warning explains that the software identification tag could not be created and included in the installation because a specific required setting was left blank. To resolve this warning, enter appropriate value in each specific setting, or select No for the Use Software Identification Tag setting.

The automation interface includes support for the new tag settings. The ISWiProject object includes a new EnableSwidtag property that lets you enable or disable the creation of a software identification tag in a project. It also includes a new SwidtagProperty property that you can use to configure various tag-related settings that are also configurable in the General Information view.

This feature applies to Basic MSI projects.

For more information, see:

- Including a Software Identification Tag for Your Product
- General Information View Settings
- ISWiProject Object

Enhancements

InstallShield includes the following enhancements.

Merge Module Projects Now Include Built-In Support for IIS, Text File Changes, and XML File Changes

Several existing views are now available in Merge Module projects in InstallShield:

- **Internet Information Services**—This view enables you to create and manage new IIS Web sites, applications, virtual directories, application pools, and Web service extensions.
- **Text File Changes**—This view lets you configure search-and-replace behavior for content in text files—for example, .txt, .htm, .xml, .config, .ini, and .sql files—that you want to modify at run time on the target system.

- **XML File Changes**—This view lets you add references for nodes and node sets of XML files that you want to change at run time.

Use these views in Merge Module projects to configure IIS, specify text file changes, and specify XML file changes. The same procedures that were used for configuring these changes in installation projects are now supported in Merge Module projects. When you build a merge module that contains project information in these views, add the merge module to an installation project, and then build a release in the installation project, InstallShield includes the applicable run-time support in the installation.

Previously, these views were available only in installation projects.

To learn more, see:

- Managing Internet Information Services
- Modifying Text Files
- Modifying XML Files

New Application Pool Identity Option for IIS 7.x

A new ApplicationPoolIdentity option is available in the Identity setting for an application pool that is configured in the Internet Information Services view. If you want a virtual identity account that is unique to the selected application pool to be used for running the application pool's worker process, select this option. Support for this option is available on target systems that have IIS 7 and later. If this new option is selected in an installation that is run on a system that has IIS 6, the NetworkService account is used instead for the application pool's identity.

This feature applies to the following project types: Basic MSI, InstallScript, InstallScript MSI, and Merge Module.

For more information, see Application Pool Settings.

Automation Interface Enhancement: New RequiredExecutionLevel Property for Specifying the Required Execution Level for Setup.exe

The read-write RequiredExecutionLevel property has been added to the ISWiRelease object. This property corresponds with the Required Execution Level setting on the Setup.exe tab for a release in the Releases view. This property is available for Basic MSI, InstallScript, and InstallScript MSI projects.

What's New in InstallShield 2011

New Features

InstallShield includes the following new features.

Unicode Support in InstallScript

InstallShield now lets you create InstallScript installations and InstallScript custom actions that support the use of Unicode in run-time strings, files, paths, registry entries, and other installation data. In addition, the InstallScript compiler and the InstallScript engine now let you pass pointers to Unicode strings to functions that are implemented outside script code; they also let you store Unicode strings in structures that are passed outside script code. Other changes in InstallScript offer additional benefits for fully supporting modern multilanguage installations.

As a result of these changes, any language displays correctly on a system that has support for it installed. End users no longer need to match the language that is used on their systems for non-Unicode programs with the language that is used in the installation. Note that the font must be installed on the target system. On some versions of Windows, the fonts for some languages are not installed by default. For example, Japanese fonts are not installed on a Windows XP English system by default; in order for an installation to use Japanese characters on such a system, the fonts would need to be installed.

Unicode Setup Launchers for InstallScript and InstallScript MSI Projects

Now all **Setup.exe** and **Update.exe** files that are built in InstallShield are Unicode. Previously, all **Setup.exe** and **Update.exe** files that were built in InstallScript and InstallScript MSI projects were ANSI.

A Unicode setup launcher can correctly display characters in the user interface of the setup launcher, regardless of whether the target system is running the appropriate code page for the language. ANSI setup launchers correctly displayed characters in the setup launcher dialogs if the target system was running the appropriate code page. However, it may have displayed garbled characters in those dialogs if the target system was not running the appropriate code page.

End users can launch **Setup.exe** and **Update.exe** files from within Unicode paths, regardless of the language on the target system. For example, end users can now launch the installation in the following path on an English system: **C:\Users***Japanese characters***\Desktop\Setup.exe**. Previously, the installation would fail.

Unicode Support for Files, Folders, Registry Entries, and Support Files

Unicode support has been added to key parts of the installation run time, enabling you to use characters from any languages simultaneously in file names, folder names, registry entries, and support file names. This enables you to install, for example, a file that has Japanese characters in its name or target path on an English system. Mixing languages works correctly regardless of the current language of the target system.

Pointer Support

The InstallScript engine and compiler now support a new pointer type called WPOINTER; wpointer and LPWSTR are equivalent names that are also available. Thus, for example, if you have a DLL function that accepts pointers to Unicode strings in its parameters, you can use this new pointer type; when the DLL function is called in the script at run time, the InstallScript engine passes a pointer to a Unicode copy of the strings instead of an ANSI version. Previously, it was possible to pass a pointer to only an ANSI copy of the strings.

Note that passing Unicode strings with the WSTRING data type that was introduced in an earlier version of InstallShield is still supported.

For more information, see:

- Pointers
- Data Types and Predefined Structures

Structure Support

Structures in InstallScript can contain any basic data type, including strings and pointers, or other structures. Now if a structure needs to contain a Unicode string and the structure is passed to an external DLL, the InstallScript engine can distinguish between string member types in that structure, and then size the structure and calculate member offsets correctly. String members that need to be stored and passed as Unicode can be declared with the WSTRING type.

Previously, if a structure needed to contain a Unicode string and the structure was passed to an external DLL, the InstallScript engine assumed that any strings in the structure were ANSI. As a result, the size of the structure and member offsets in the structure could have been wrong, causing the DLL to incorrectly read or write data related to the structure. Attempting to use WSTRING for string members of a structure did not have any effect.

You can leave existing strings as STRING types; the InstallScript engine will continue to treat these as ANSI strings when passing them outside of script code.

Pointer members in structures can also now be declared as WPOINTER. This enables you to store pointers to Unicode strings in a structure.

For more information, see:

- Data Structures
- Data Types and Predefined Structures

String Table Support

InstallShield now uses Unicode encoding to build the string tables for InstallScript projects. Because of this support, string tables for InstallScript projects can now contain multiple languages, and they are not affected by a target system's code page settings. In addition, string tables can now include strings for languages—such as Hindi—that have no code page.

It is recommended that any strings that will be displayed in the user interface of an installation be stored in the string table. Although strings can be written directly in InstallScript code, they are not stored as Unicode; thus, they are displayed correctly only when the installation is run on systems that have the correct code page. Storing strings in the string table and referencing them from InstallScript code eliminates this issue.

For more information, see:

- Using String Entries in InstallShield
- String Constant Operator (@)

Unicode InstallScript Dialogs

InstallScript dialogs now include support for Unicode. This enables you to mix, for example, Japanese and German, or Russian and Polish, on InstallScript dialogs. These mixed languages work correctly regardless of the current language of the target system.

This feature is available in the following project types: InstallScript and InstallScript MSI.

Unicode Support in the InstallScript Debugger

The InstallScript Debugger now has support for Unicode. Thus, for example, if you are debugging a line of InstallScript code such as szMsg = @ID_MSG, you can now see the value in szMsg, regardless of what language operating system you are using and what language is used for the strings. If the value of ID_MSG contains Chinese characters or characters from mixed languages, you can now see the appropriate characters instead of question marks in various areas of the InstallScript Debugger: in the variable window, in the watch window, and in the tooltip that appears for the variable in the script window.

For information on the InstallScript Debugger, see Debugging InstallScript-Based Installations.

Unicode Support in InstallShield Cabinet and Log File Viewer

The InstallShield Cabinet File Viewer and the InstallShield Log File Viewer have been combined into one new tool—the InstallShield Cabinet and Log File Viewer—which lets you open and review InstallScript cabinet files (.cab), InstallScript header files (.hdr), and InstallScript log files (.ilg). This tool now includes Unicode support. Thus, the tool can correctly display the characters in each file name, registry key, shortcut, and other data in the .cab, .hdr, or .ilg file, regardless of what language operating system you are using and what language is used for the strings. Previously, in some cases, some of the characters were displayed as question marks.

The cabinet and header file support applies to the following project types: InstallScript and InstallScript Object.

The log file support applies to the following project types: InstallScript and InstallScript MSI.

To learn about this tool, see:

- InstallShield Cabinet and Log File Viewer
- Viewing InstallScript Cabinet Files (.cab) and Header Files (.hdr)
- Viewing InstallScript Log Files (.ilg)

Integration with Team Foundation Server (TFS)

InstallShield has enhanced support for integrating with Team Foundation Server (TFS) 2010.

Now when you are using InstallShield from within Visual Studio 2010, you can access the Source Control Explorer to integrate your InstallShield project with Team Foundation version control and manage changes to your InstallShield projects and your Visual Studio solutions.

You can also use Team Foundation Build to compile, test, and deploy your InstallShield projects and your Visual Studio solutions on a regular basis, or on demand. Your installation is automatically updated with your latest source files every time that your solution is built.

In addition, if you install Team Explorer on the same machine that has InstallShield and Visual Studio, you can use Team Explorer from within your InstallShield projects that are open in Visual Studio. This enables you to perform tasks such as the following:

- Use Source Control Explorer when you are working on your InstallShield projects.
- Configure builds for your InstallShield projects and Visual Studio solutions.
- Queue new builds.
- Track work items such as bugs and tasks for your InstallShield projects and your Visual Studio solutions.

For more information, see:

- Integrating with Microsoft Visual Studio Team Foundation Server
- Adding InstallShield Projects to Team Explorer

Microsoft SQL Server 2008 R2 Support

InstallShield now includes support for running SQL script on SQL Server 2008 R2. In addition, InstallShield includes SQL Server 2008 R2 in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports.

This feature is available in the following project types: Basic MSI, InstallScript, and InstallScript MSI.

New InstallShield Prerequisites for SQL Server 2008 R2 Express, SQL Server Native Client, Visual C++ 2010, and Other Redistributables

InstallShield includes a number of new InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Microsoft SQL Server 2008 R2 Express
- Microsoft SQL Server 2008 Native Client 10.00.2531
- Microsoft SQL Server Native Client 9.00.4035
- Microsoft SQL Server System CLR Types 10.00.2531
- Microsoft SQL Server 2008 Management Objects 10.00.2531
- Microsoft Visual C++ 2010 Redistributable Package
- Microsoft Visual C++ 2008 SP1 Redistributable Package
- Microsoft Visual C++ 2005 SP1 Redistributable Package (x64)
- Windows Installer 3.1 - Japanese
- MSXML 6.0 SP1 - Japanese
- Microsoft .NET Framework 4.0 Client Japanese Language Pack
- Microsoft .NET Framework 4.0 Full Japanese Language Pack

Improved Script Editors for Creating and Modifying InstallScript Code, SQL Scripts, VBScript Custom Actions, and JScript Custom Actions

Several views in InstallShield have an improved script editor that you can use to write code for your projects. The script editors include the following improvements:

- **Expanded auto completion**—When you are typing in a script editor, InstallShield displays a pop-up list of alphabetically ordered functions, keywords, and constants that begin with the letters that you are typing. Instead of manually typing the entire word, you can select it in the pop-up list, and InstallShield adds it to your script.

  If you are using the script editor in the InstallScript view, you can enter the string constant operator (@) to see a pop-up list of the available string identifiers. You can select the appropriate one in the list, and InstallShield adds it to your script.

  If auto completion for local variables is also enabled, the pop-up list in the script editor of the InstallScript view also contains local variables.

  Auto completion can increase your efficiency because it can reduce the time that you spend typing code. It can also help you avoid typographical errors in your code.

  The script editor in the InstallScript view of earlier versions of InstallShield is the only script editor that provided some auto completion support; thus, auto completion functionality was not available for SQL scripts, VBScript code, or JScript code. In addition, the available pop-up list was limited to InstallScript functions; none of the keywords, constants, local variables, or string identifiers were available for auto completion.

- **Detailed InstallScript Function Call Tips**—If function call tips are enabled, InstallShield shows InstallScript function call tips—a type of tooltip—when you are entering function calls in your script in the InstallScript view. A function call tip shows a built-in function's parameter information. It also shows a description of the built-in

function, as well as a description of the parameter that you are entering. The detailed call tips enable you to see help information about a function without switching from the script editor to the help library, and then back to the script editor.

The InstallScript view in earlier versions of InstallShield included support for call tips, but the call tips did not display as much information. The call tips showed a function call with all of the function's parameters; however, the call tips did not include a description of the function, or of the parameter that you are entering.

- **Syntax folding**—InstallShield now lets you specify whether you want the script editors in various views in InstallShield to include support for syntax folding. When syntax folding is enabled, InstallShield adds a plus sign (+) or a minus sign (–) in the margin next to each line of code that starts an expandable or collapsible block of script. You can click a plus sign to expand hidden code, or a minus sign to hide visible code.

  Syntax folding can help you minimize the clutter of large scripts and focus on the code that is relevant to the work that you are currently doing. It can also help you see the overall structure of a script.

The views that contain the improved script editor are the InstallScript view, the SQL Scripts view, and the Custom Actions and Sequences view (when you are viewing a VBScript or JScript file in this view).

By default, syntax folding is disabled in the script editors; auto completion, local variable auto completion, and function call tips are enabled. To enable or disable any of the aforementioned functionality, use the Script Editor Properties dialog box. This dialog box also lets you modify other settings in the script editors, such as font, syntax colors, and line numbering. To access this dialog box, right-click in a script editor and then click Properties.

For more information, see the following:

- Working with the Script Editor Pane in Various Views
- Using Auto Completion when Writing Code in the Script Editors
- Viewing Function Call Tips for an InstallScript Function in the Script Editor
- Changing Colors for Syntax Highlighting in the Script Editors
- Enabling or Disabling Syntax Folding in the Script Editors
- Keyboard Shortcuts for the Script Editors
- Script Editor Properties Dialog Box

Ability to Create 64-Bit Microsoft App-V Packages

The Microsoft App-V Assistant in InstallShield includes support for converting a 64-bit Windows Installer package into a 64-bit Microsoft App-V package, which can be deployed on 64-bit Windows systems with the 64-bit Microsoft App-V 4.6 client. This, in combination with the new 64-bit repackaging support that is available with the Repackager, enables you to convert any 64-bit installation into a 64-bit App-V package.

It is recommended that you perform the conversion of 64-bit Windows Installer packages to App-V packages on a 64-bit Windows system. If you attempt the conversion on a 32-bit system, it could result in a failure to extract COM information for 64-bit binaries. Also, in some cases, Windows Installer packages contain shortcuts that target executable files that are not found in the package itself. If these shortcuts target executable files are found in 64-bit locations, these shortcuts are not handled correctly on 32-bit systems.

The Microsoft App-V Assistant is available if you purchase the Premier Edition of InstallShield.

Ability to Specify a Search Path for InstallShield Prerequisites; Path Variable and Relative Path Support for Source Locations of Prerequisite Files

InstallShield now lets you specify the folders where InstallShield should search for InstallShield prerequisite files (.prq files). This functionality makes it easier for teams of users to share InstallShield prerequisites with each other, and for storing prerequisites in source code control. Previously, InstallShield searched for .prq files in the following location only: *InstallShield Program Files Folder***SetupPrerequisites**.

InstallShield offers several ways for specifying the folders:

- If you are editing or building from within InstallShield, use the new Prerequisites tab on the Options dialog box—which is displayed when you click Options on the Tools menu—to specify a comma-delimited list of machine-wide folders and current-user folders. This tab is similar to the Merge Modules tab on the Options dialog box, which lets you specify search paths for merge modules.
- If you are building from the command line with **ISCmdBld.exe**, use the new `-prqpath` parameter to specify a comma-delimited list of folders.

  If you use an .ini file to specify **ISCmdBld.exe** parameters, you can use the new PrerequisitePath parameter in the [Mode] section of your .ini file to specify a comma-delimited list of folders.
- If you are building through MSBuild or Team Foundation Server (TFS), use the new PrerequisitePath parameter on the InstallShield task. This parameter is exposed as the ItemGroup InstallShieldPrerequisitePath when the default targets file is used. To specify multiple paths, use an ordered array of paths.

When you are using the Files to Include tab in the InstallShield Prerequisite Editor to add files to an InstallShield prerequisite, the editor now uses predefined path variables such as <WindowsFolder> and <ISProductFolder> if appropriate. In addition, if the files that you are adding are stored in the same folder as the InstallShield prerequisite's .prq file—or a subfolder of the folder that contains the .prq file—the InstallShield Prerequisite Editor uses a relative path for the file in the .prq file. Note that if you view the file's path on the Files to Include tab, the InstallShield Prerequisite Editor lists the full path, not the relative path.

Note that if you use the Save As command in the InstallShield Prerequisite Editor to change the location of a .prq file, the InstallShield Prerequisite Editor updates the paths of the prerequisite's files if appropriate.

To learn more, see:

- Specifying the Directories that Contain InstallShield Prerequisites
- Prerequisites Tab
- ISCmdBld.exe
- Passing Command-Line Build Parameters in an .ini File
- Microsoft Build Engine (MSBuild)

New Hyperlink Control for Windows Installer–Based Dialogs

The Controls toolbar includes a new hyperlink control that you can use in the Dialogs view for Windows Installer–based dialogs. The hyperlink control displays an HTML link; clicking the link at run time opens a page in the default browser on the target system.

Windows Installer 5 includes support for this new hyperlink control. If this control is used on a dialog that is displayed on a system that has an earlier version of Windows Installer, run-time error 2885 occurs, and the installation aborts. Therefore, if you want to use the hyperlink control on a dialog but your installation targets

systems that have Windows Installer 4.5 or earlier, it is recommended that you include two versions of the dialog in your project: one with the hyperlink control, and one without it. Add conditions to the dialogs to show or hide them, depending on the version of Windows Installer that is present.

This feature is available in the following project types: Basic MSI and Merge Module.

To learn more, see Hyperlink Control.

Ability to Specify a Custom Icon and Custom Version Resource Properties for Setup.exe and Update.exe

InstallShield now lets you use a custom icon and custom version resource properties for **Setup.exe** files that you create at build time. The icon and the version resource properties are displayed on the Properties dialog box for **Setup.exe**; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. End users can also see the icon when they view your **Setup.exe** file in Windows Explorer. This support is available in Basic MSI, InstallScript, and InstallScript MSI projects.

The same functionality (the ability to specify a custom icon and custom version resource properties) is also now available for **Update.exe** files that you create in Basic MSI, InstallScript MSI, and QuickPatch projects.

Custom Icon for Setup.exe and Update.exe

The Setup.exe tab in the Releases view has a Setup.exe Icon File setting that lets you specify the icon that should be used for your **Setup.exe** setup launcher. The icon can be in an .exe, .dll, or .ico file. If you do not specify an icon, InstallShield uses a default icon for your **Setup.exe** file.

Previously, support for specifying a custom **Setup.exe** icon was available for InstallScript projects if the **Setup.exe** file that you were building was a self-extracting, single-file setup launcher. The support was not available in Basic MSI or InstallScript MSI projects. In addition, it was not available for uncompressed InstallScript installations.

A new Icon setting lets you specify the icon that should be used for your **Update.exe** launcher. This setting is available on the Advanced tab for a patch configuration in the Patch Design view of Basic MSI and InstallScript MSI projects. It is also available on the Advanced tab of the Build Settings area in the General Information view of QuickPatch projects. If you do not specify an icon, InstallShield uses a default icon for your **Update.exe** file.

Previously, InstallShield did not include any support for specifying a custom icon for **Update.exe** files.

Custom Version Resource Properties for Setup.exe and Update.exe

When InstallShield is configuring the following version resources of your **Setup.exe** setup launcher at build time, it now uses the custom information that you entered in the General Information view and the Releases view:

- Company name
- Product name
- Product version
- Copyright
- File version
- File description

To use a custom copyright notice and a custom file description, ensure that you select Yes in the Use Custom Version Properties setting for the release in the Releases view.

Previously, InstallShield did not use any custom information in many cases. For example, the InstallScript **Setup.exe** files that InstallShield previously created contained details that pertained to the version of InstallShield that was used to build the **Setup.exe** file. Thus, the product name that was displayed in the **Setup.exe** Properties dialog box was InstallShield, rather than the name of your product.

Several new settings—Company Name, Product Name, Product Version, Description, and Copyright—let you specify the custom information that you want InstallShield to use when you build an **Update.exe** file. These settings are available on the Advanced tab for a patch configuration in the Patch Design view of Basic MSI and InstallScript MSI projects.They are also available on the Advanced tab of the Build Settings area in the General Information view of QuickPatch projects.

Previously, InstallShield did not use any custom version resource information for **Update.exe** files.

To learn more, see:

- Specifying the Icon for the Setup Launcher
- Customizing File Properties for the Setup Launcher
- Setup.exe Tab for a Release
- Specifying the Icon for the Update Launcher
- Customizing File Properties for the Update Launcher
- Advanced Tab (in the Patch Design view)
- Advanced Tab (for a QuickPatch project)

Ability to Specify Commands that Run Before, During, and After Builds

The Premier edition of InstallShield includes new release settings that you can use to specify commands that you want to be run at various stages of the build process. These new settings are available on a new Events tab when you select a release in the Releases view:

- **Prebuild Event**—Use this setting to specify the command that you want to be run before InstallShield starts building the release. This event runs after InstallShield creates the release folder and log file, but before InstallShield starts building the release.

  This setting is available in the following project types: Basic MSI, InstallScript, InstallScript MSI, and Merge Module.

- **Precompression Event**—Use this setting to specify the command that you want to be run after InstallShield has built the .msi package and the .cab files (if your product's data files are to be stored in .cab files). Note that this event occurs after .cab files are streamed into the .msi package, but before the .msi package has been digitally signed and streamed into the **Setup.exe** file.

  This setting is available in the following project types: Basic MSI and InstallScript MSI.

- **Postbuild Event**—Use this setting to specify the command that you want to be run after InstallShield has built and signed the release.

  This setting is available in the following project types: Basic MSI, InstallScript, InstallScript MSI, and Merge Module.

The new Events tab replaces the previously available Postbuild tab. Note that the settings that were previously available on the Postbuild tab are now displayed on the new Events tab. In addition, the publish-related settings that were previously available on the Build tab in Merge Module projects have been moved to the Events tab.

The automation interface now includes support for the new build event settings. The ISWiRelease object includes three new properties that let you specify commands for various stages of the build process:

- PrebuildEvent
- PrecompressionEvent
- PostbuildEvent

To learn more, see:

- Specifying Commands that Run Before, During, and After Builds
- Events Tab for a Release
- ISWiRelease Object

Ability to Set an Expiration Date for Setup.exe

InstallShield now lets you set an expiration date, as well as an expiration message, for **Setup.exe**. If end users try to run **Setup.exe** on or after the date that you have specified in your project, the expiration message is displayed, and the installation exits.

To set an expiration date and a message for your **Setup.exe** file, use the new Expiration Date and Expiration Message settings on the Setup.exe tab for a selected release in the Releases view. By default, no expiration date is configured for **Setup.exe**. If you specify an expiration date, you can use the default expiration message, or specify your own custom message.

The automation interface now includes support for these new settings. The ISWiRelease object includes two new properties that let you set the expiration date and message: ExpirationDate and ExpirationMessage.

This feature is available in the following project types: Basic MSI and InstallScript MSI.

For more information, see:

- Setup.exe Tab for a Release
- ISWiRelease Object

Ability to Import Visual Studio Setup and Merge Module Projects into Existing InstallShield Projects; Improvements for the Project Converter

InstallShield now lets you import a Visual Studio setup project (.vdproj) into an InstallShield Basic MSI or Merge Module project (.ism), or import a Visual Studio merge module project (.vdproj) into an InstallShield Basic MSI or Merge Module project (.ism). These tasks enable you to develop InstallShield installation and merge module projects that contain the same data and settings that were in your Visual Studio project. The wizard imports the project outputs, files, registry keys, file extensions, custom actions, target system searches, and launch conditions from your Visual Studio project into your existing InstallShield project.

To import a Visual Studio project into an existing InstallShield project, use the new Visual Studio Deployment Project Import Wizard in InstallShield. The wizard lets you choose whether to import or ignore certain settings in the Visual Studio project.

The existing support for converting a Visual Studio project into a new InstallShield project has been expanded. If your Visual Studio project contains predefined prerequisites, InstallShield now converts them to equivalent InstallShield prerequisites during the conversion process. This same prerequisite conversion functionality is available if you use the new wizard to import a Visual Studio project into an InstallShield project.

If your Visual Studio project contains one or more project outputs, use the import wizard instead of the conversion process. The InstallShield project must be in the same Visual Studio solution that contains the Visual Studio setup or merge module project and all of its project dependencies. Note that you must be using InstallShield from within Visual Studio when you use the import wizard in order for the wizard to import the project outputs into your InstallShield project.

To learn more, see the following:

- Converting or Importing Visual Studio Projects into InstallShield Projects
- Visual Studio Deployment Project Import Wizard

Unicode and UTF-8 Support for SQL Scripts

InstallShield now has design-time and run-time support for SQL scripts that have either Unicode BOM encoding or UTF-8 BOM encoding. You can use the SQL Scripts view to add SQL scripts with either of these encoding types to your project. You can also edit those SQL scripts from within that view as necessary. At run time, the SQL scripts are executed during installation and uninstallation when needed.

Previously, InstallShield had run-time support but not design-time support for SQL scripts with Unicode BOM encoding. Thus, when you added a script with this encoding to the SQL Scripts view of your project, InstallShield prompted you to specify whether you wanted to convert the script to ANSI format. If you allowed InstallShield to convert it to ANSI, you could edit it from within the SQL Scripts view. However, in some cases, it might have displayed and used garbled characters at design time and at run time. If you did not allow InstallShield to convert it to ANSI, the file remained with Unicode BOM encoding. Although this encoding allowed your installation to properly use strings in languages that did not match the code page of the target system at run time, you could not view or edit the script from within InstallShield.

In addition, InstallShield previously did not have adequate support for SQL scripts with UTF-8 BOM encoding. If you added a SQL script with this encoding type to your project and the script contained strings in a language that did not match the code page of the development system, the SQL Scripts view treated the file as an ANSI file, and it may have displayed garbled characters. In addition, garbled characters were used when your SQL script was executed at run time. Also, the byte order mark was represented as garbled characters.

If you create a new SQL script from within the SQL Scripts view, InstallShield uses Unicode BOM encoding for the file. If you prefer to use ANSI or UTF-8 BOM encoding for the SQL scripts view, it is recommended that you create an .sql file with the proper encoding in a different tool, and then import or insert the script in the SQL Scripts view of your project.

This feature is available in the following project types: Basic MSI, InstallScript, and InstallScript MSI.

Predefined System Searches for SQL Server 2008 Express SP1 and Adobe Reader 9

InstallShield has new predefined system searches:

- SQL Server 2008 Express SP1
- Adobe Reader 9

If your installation requires one or both of these, you can use the System Search view or the Installation Requirements page in the Project Assistant to add these system searches to your project. When end users launch your installation, Windows Installer checks the target system to see if the requirements are met; if they are not met, the installation displays the error message that is defined for the system search.

This feature applies to Basic MSI and InstallScript MSI projects.

Support for 64-Bit Managed-Code Custom Actions

InstallShield now has support for 64-bit managed-code custom actions. When you build a release that includes a managed-code custom action in your project, InstallShield attempts to determine the target architecture (32 bit or 64 bit) of the main .NET assembly that is associated with the custom action. InstallShield configures the release so that the appropriate version of the .NET Framework—32 bit or 64 bit—is used to run your managed code at run time.

If you want to override the new default behavior, use the Direct Editor view to add a new record with the following fields to the **ISClrWrap** table.

- **Action_**—Indicate the name of the managed-code custom action that you want to modify.
- **Name**—Enter the following: **TargetPlatform**
- **Value**—Specify the appropriate architecture. To use a 32-bit version of the .NET Framework, enter the following: **x86**

  To use a 64-bit version of the .NET Framework, enter the following: **x64**

This feature is available in the following project types: Basic MSI, InstallScript MSI, and Merge Module.

To learn more, see Using 32-Bit vs. 64-Bit Managed-Code Custom Actions.

Support for 64-Bit .NET Installer Classes and COM Interop

InstallShield now has support for 64-bit .NET installer classes and COM interop. If you are using InstallShield on a 64-bit system, the .NET tab on the Options dialog box—which is displayed when you click Options on the Tools menu in InstallShield—now lets you specify different paths for the 32-bit and 64-bit locations of the **Regasm.exe** and **InstallUtilLib.dll** files that are included with the .NET Framework. InstallShield uses the paths that you specify at build time for releases that include .NET installer classes and COM interop.

If you are building from the command line with **ISCmdBld.exe** and you use the existing `-t` parameter to specify the path of the 32-bit version of the .NET Framework, **ISCmdBld.exe** now uses the 64-bit location of **Regasm.exe** and **InstallUtilLib.dll** when appropriate.

If you are building through MSBuild or Team Foundation Server (TFS) and you use the existing DotNetUtilPath parameter on the InstallShield task to specify the path of the 32-bit version of the .NET Framework, the build now uses the 64-bit location of **Regasm.exe** and **InstallUtilLib.dll** when appropriate.

To learn more, see:

- .NET Tab (on the Options dialog box)
- ISCmdBld.exe
- Microsoft Build Engine (MSBuild)

Support for Using the InstallScript Function DotNetCoCreateObject or Managed-Code Custom Actions with DLLs that Target .NET Framework 4

If you create a DLL in Visual Studio 2010 and the DLL uses .NET Framework 4, you can now use the InstallScript function **DotNetCoCreateObject** to call functions in this DLL. Previously, the installation crashed if the DLL used version 4 of the .NET Framework, but not with earlier versions. This feature is available in the following project types: InstallScript and InstallScript MSI; it is also available in Basic MSI and Merge Module projects with InstallScript custom actions.

In addition, you can now use the same sort of DLL file in a managed-code custom action. Previously, the installation crashed if it contained a managed-code custom action for a DLL that used version 4 of the .NET Framework. This feature is available in the following project types: Basic MSI, InstallScript MSI, Merge Module.

For more information, see:

- DotNetCoCreateObject
- Using 32-Bit vs. 64-Bit Managed-Code Custom Actions

Ability to Override the Values of Path Variables for a Release

InstallShield now lets you override the values of your project's custom path variables (standard path variables, environment path variables, and registry path variables) for each release in your project. This functionality enables you to essentially replace certain files and folders in your project with others at build time, depending on the particular release that you are building.

For example, you might use this functionality to swap out the binaries for custom actions. If you have set up separate releases for 32-bit and 64-bit target systems, you can override the path variable that is used to refer to the DLL that is selected for a custom action. Then InstallShield could include a 32-bit DLL for your 32-bit release and a 64-bit DLL for your 64-bit release. Note that overriding path variables to swap out files that your installation is installing is not recommended. This is because you should use separate components for 32-bit and 64-bit versions of a file.

To override one or more path variables in your project, use the new Path Variables Overrides setting, which is on the Build tab for a release in the Releases view. Note that if you override a path variable both in the new Path Variables Overrides setting, and with the `-l` parameter to **IsCmdBld.exe** or through MSBuild, the command line or MSBuild value takes precedence over the value that is set in the release setting.

The automation interface now includes support for this new setting. The ISWiRelease object includes a new PathVariableOverrides property that lets you override the values of your project's user-defined path variables, environment variables, and registry variables for a release.

This feature is available in the following project types: Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module.

To learn more, see the following:

- Overriding the Value of a Custom Path Variable for a Release
- Path Variable Overrides setting
- Path Variable Overrides Dialog Box
- ISWiRelease Object

Ability to Configure MIME Types for IIS Web Sites, Applications, and Virtual Directories

The Internet Information Services view has a new MIME Types setting that you can configure for a Web site, application, or virtual directory in your project. This setting lets you identify the types of content that can be served from the Web server on the target system to a browser or mail client.

This feature applies to the following project types: Basic MSI, InstallScript, and InstallScript MSI.

For more information, see:

- MIME Types setting for a Web site

- MIME Types setting for an application or virtual directory

Ability to Overwrite Existing IIS Application Pool or Only Create It If It Does Not Exist

The Internet Information Services view has a new Overwrite Existing Application Pool setting. This setting is displayed in the right pane when you select an application pool in the Internet Information Services view. This setting lets you specify the behavior that you want to occur if the selected application pool already exists on the target system at run time: the installation can either overwrite the application pool's settings on the target system, or leave the application pool as is. The default value for this setting is Yes, indicating the an existing application pool is overwritten at run time.

This feature applies to the following project types: Basic MSI, InstallScript, and InstallScript MSI.

For more information, see the description of the Overwrite Existing Application Pool setting for an application pool.

New Billboard Style in InstallScript and InstallScript MSI Projects

InstallScript and InstallScript MSI projects include support for a new style of billboard that is displayed above the progress bar on the progress dialog. This style of billboard supports Adobe Flash application files (.swf), as well as image files (.bmp, .gif, .jpg, and .jpeg).

If a project contains both Flash files and image files but a target system does not have the Adobe Flash Player, the installation can detect that and display image billboards in place of the Flash billboard.

Previously, the only available billboard support in InstallScript and InstallScript MSI projects required the use of a background window. The new billboard style does not require a background window.

To add the new style of billboard to a project, use the Support Files/Billboards view to add billboard files to your project. To display this new style of billboard at run time, use the new STATUSBBRD constant with the **Enable** function.

Note that the new style of billboard is not available in projects that use skinned dialogs.

To learn more, see:

- Displaying Billboards in InstallScript and InstallScript MSI Installations
- Billboard Styles and File Types for InstallScript and InstallScript MSI Projects
- Adding or Modifying the Code in an InstallScript or InstallScript MSI Project to Display Billboards
- Adding an Adobe Flash Application File Billboard to an InstallScript or InstallScript MSI Project
- Adding an Image Billboard to an InstallScript or InstallScript MSI Project
- Naming Billboard Files in an InstallScript or InstallScript MSI Project

Support for Looping Through Image Billboards in Basic MSI Projects

The Billboards view in Basic MSI projects has a new Loop Billboards setting that lets you specify whether you want your installation to continuously loop the image billboards until the file transfer completes and the installation displays the appropriate SetupComplete dialog.

If you select Yes for this setting and the file transfer takes more time than you have allocated for the billboards, the installation restarts the display of billboards from the beginning. The loop continues, if necessary, until the file transfer ends. The default value for this setting is No, which matches the behavior in earlier versions of InstallShield.

Previously, if the file transfer took more time than what you had allocated for the billboards, the installation continued to display the last billboard until the file transfer finished; it did not loop the billboards.

For more information, see:

- Run-Time Behavior of a Basic MSI Installation that Includes Billboards
- Billboard Settings

Windows Installer 5 Support for Configuring Windows Service Permissions

InstallShield now includes support for configuring permissions for Windows services; this support uses the **MsiLockPermissionsEx** table that was introduced with Windows Installer 5. Windows Installer 5 supports the new service permission settings; earlier versions of Windows Installer ignore them.

To configure the new permission settings for a service, use the Services node in the Advanced Settings area for a component in the Setup Design view (in installation projects) or the Components view. The Services node is where you add new services to your project. When you select a subnode under the Services node, you can configure the settings—including the new permission-related settings—of that service in the right pane.

This functionality is available in the following project types: Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

For more information, see:

- Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment
- Services Settings

Automation Interface Support for Configuring Dynamic File Links in InstallScript Projects

The ISWiDynamicFileLinking object and the ISWiDynamicFileLinkings collection of the automation interface now include support for dynamic file links in InstallScript projects. In addition, two methods and a property of the ISWiComponent object are now applicable to InstallScript projects: the AddDynamicFileLinking method lets you add a new dynamic file link to a component, the RemoveDynamicFileLinking method lets you remove a component's dynamic file links, and the ISWiDynamicFileLinkings property lets you get a component's collection of dynamic file links. The read-only DynamicFile property for the ISWiFile object is also now applicable to InstallScript projects: use this property to determine whether the file's source is linked to the component dynamically or statically.

For more details, see:

- ISWiDynamicFileLinking Object
- ISWiDynamicFileLinkings Collection
- ISWiComponent Object
- AddDynamicFileLinking Method
- RemoveDynamicFileLinking Method
- ISWiFile Object

New FlexNet Connect 12.01 Redistributables Available

InstallShield includes support for FlexNet Connect 12.01 in Basic MSI and InstallScript MSI projects. Use the Update Notifications view in InstallShield to include one of the two FlexNet Connect 12.01 merge modules—one has the Common Software Manager, and the other does not.

Enhancements

InstallShield includes the following enhancements.

Unicode Views in InstallShield

Several views and areas in InstallShield have been enhanced to display and allow you to enter characters from all languages. For example, now you can use Chinese characters in file names, paths, and registry entries if you are configuring system searches in the System Search view on an English machine. Previously, the characters were displayed in these areas of InstallShield as question marks.

The areas in InstallShield that have been enhanced to include the Unicode support are the System Search view, the tabs for a release in the Releases view, and the Multiple Instances tab for a release in the Releases view. Note that Unicode support was previously introduced in many other views in InstallShield 2010.

The improvements for the System Search view are applicable to the following projects: Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

The improvements for the tabs for a release in the Releases view are applicable to the following projects: Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module.

The improvements for the Multiple Instances tab in the Releases view are applicable to Basic MSI projects.

Command-Line Support for Generating a Log File While Using InstallShield MSI Diff

InstallShield MSI Diff includes a new `/out` command-line parameter. You can use this new parameter while running InstallShield MSI Diff from the command line to generate a log file that identifies the differences between two .msi, .msm, or .pcp files, or that shows the differences that are applied to a Windows Installer database by a transform (.mst) or patch file (.msp). You can also use this tool from the command line to generate a log file that identifies the differences between two InstallShield project files (.ism or .ise) that are saved in binary format.

To learn more, see Running InstallShield MSI Diff from the Command Line to Generate a Log File of Differences.

Support for Listing Local Microsoft SQL Server Instances on 64-Bit Systems

If you add SQL support to a project through the SQL Scripts view in InstallShield and the installation is run on a 64-bit target system, the built-in SQL-related run-time dialogs now list 64-bit local SQL Server instances, as well as 32-bit local SQL Server instances, 32-bit network SQL Server instances, and 64-bit network SQL Server instances. Previously when the installation was run on a 64-bit target system, the built-in SQL-related run-time dialogs did not list any 64-bit local SQL Server instances; they listed only 32-bit local instances, 32-bit network instances, and 64-bit network instances.

This functionality is available in the following project types: Basic MSI, InstallScript, and InstallScript MSI.

InstallScript Logging and Uninstallation of 64-Bit Registry Changes

By default, the InstallScript engine now logs the changes that it makes to the 64-bit part of the registry during an InstallScript or InstallScript MSI installation on 64-bit target systems. Furthermore, the 64-bit registry changes that are logged by the InstallScript engine are now uninstalled during uninstallation.

Previously, if the InstallScript engine made changes to the 64-bit part of the registry during an installation, those changes were not logged as 64 bit specific, so they were not removed during uninstallation.

Support for Microsoft SQL Server Management Studio

The SQL Scripts view now includes support for Microsoft SQL Server Management Studio. To open a SQL script file of your project in SQL Server Management Studio, right-click the SQL script file in the SQL Scripts view and then click Open Script in Microsoft SQL Server Management Studio. This new command replaces the Open Script in Microsoft Query Analyzer command on the same context menu.

This functionality is available in the following project types: Basic MSI, InstallScript, and InstallScript MSI.

Support for Adding Project Outputs from Visual Studio Web Setup Projects

If you create a Visual Studio solution that includes a Web setup project and an InstallShield installation project and you are using InstallShield from within Visual Studio, you can now add project outputs from the Web setup project to your InstallShield project.

Ability to Select a Windows Installer Property when Configuring a Text File Change

Two settings in the Text File Changes view have been enhanced to make it easier to configure search-and-replace behavior for content in text files. The Find What and Replace What settings, which are displayed in the right pane when you select a replacement node in the Text File Changes view, now let you select a Windows Installer property from a list, or type a string. The list consists of all of the properties that are available in the Property Manager view of your project. Previously, you could manually enter a string or a property in these settings, but the list of properties was not available.

This enhancement is available in the following project types: Basic MSI and InstallScript MSI.

InstallScript Text Substitution Improvements for Resolving Environment Variable Paths in IISROOTFOLDER

InstallScript installations now have support for installing IIS data to IISROOTFOLDER if the IIS PathWWWRoot registry value contains an environment variable. Previously, when text substitution was performed for the IISROOTFOLDER variable in an InstallScript installation that included IIS support, the environment variable part of the path of IISROOTFOLDER was not resolved on Windows Server 2008 and later systems. If any components were configured to be installed to IISROOTFOLDER, they failed to install because the unresolved path was an invalid path.

Ability to Specify the Required Execution Level for Update.exe Manifests

The Required Execution Level setting is now available on the Advanced tab for a patch configuration in the Patch Design view of Basic MSI and InstallScript MSI projects. It is also available on the Advanced tab of the Build Settings area in the General Information view of QuickPatch projects.

Use these new settings to specify the minimum execution level that is required by your installation's **Update.exe** file for running the upgrade on Windows Vista and later platforms. InstallShield adds a manifest that specifies the required level. By default, InstallShield uses the level that was configured in the previous setup launcher's manifest.

Previously, the Required Execution Level setting was available only for the **Setup.exe** setup launchers. If you created an **Update.exe** patch, InstallShield used the same required execution level that was configured in the previous setup launcher's manifest.

To learn more, see:

- Advanced Tab (in the Patch Design view)
- Advanced Tab (for a QuickPatch project)

Ability to Create .cab Files Without Compression

A new Cab Optimization Type setting is available on the Build tab for a release that is selected in the Releases view. If Compressed or one of the custom options is selected for the Compression setting, use the Cab Optimization Type setting to specify the type of compression that InstallShield should use when building the release's .cab files. The available options are LZX compression, MSZIP compression, or no compression.

The Cab Optimization Type setting replaces the Optimize Size setting, which had support for only LZX compression and MSZIP compression; it did not let you skip compression.

The automation interface now includes support for this new setting. The ISWiRelease object includes a new CabCompressionType property that lets you specify which of the three compression options you want to use when you build a release through the automation interface.

This enhancement is available in the following project types: Basic MSI and InstallScript MSI.

For more information, see Build Tab for a Release.

Expanded List of Predefined Operating System Conditions in the InstallShield Prerequisite Editor

The Prerequisite Condition dialog box, which is displayed when you are adding or modifying a condition for an InstallShield prerequisite in the InstallShield Prerequisite Editor, now has predefined operating system conditions for Windows Server 2008 R2. Some of the Windows 7 options have been renamed to make it more clear which versions of Windows are checked, since some of the options check for the workstation version, the server version, or either version. The Prerequisite Condition dialog box also has new options that check only for Windows 7, not Windows Server 2008 R2. These changes let you quickly define more-targeted operating system conditions for any InstallShield prerequisites.

You can add InstallShield prerequisites to the following project types: Basic MSI, InstallScript, and InstallScript MSI.

New Refresh Button in the Redistributables and Prerequisites Views

The Redistributables view (in Basic MSI, InstallScript MSI, and Transform projects) and the Prerequisites view (in InstallScript projects) contains a new Refresh button that you can use to refresh the list of redistributables that are displayed in the view. Previously, if you added redistributables to your machine when either view was open in InstallShield, it was necessary to close and then reopen the project in order to see an updated list in the view.

This enhancement is available in the following project types: Basic MSI, InstallScript, InstallScript MSI, and Transform.

Support for Setting Permissions for the IUSR User Account

InstallShield now has support for securing files, folders, and registry keys for the well-known user account IUSR.

If you are using the custom InstallShield handling method of configuring permissions in your project, the User list on the Permissions dialog box has a new IUSR option. To use the custom InstallShield handling method, select this option in the Locked-Down Permissions setting in the General Information view. This functionality is available in the following project types: Basic MSI, InstallScript MSI, Merge Module, and Transform.

If you are using the InstallScript function **SetObjectPermissions** to set permissions, you can now pass IUSR for the szUser parameter. This functionality is available in InstallScript events in the following project types: InstallScript and InstallScript MSI. This functionality is also available through InstallScript custom actions in Basic MSI, InstallScript MSI, and Merge Module projects.

New IS_PERMISSIONS_OPTION_ALLOW_ACCESS Constant for the SetObjectPermissions Function

A new IS_PERMISSIONS_OPTION_ALLOW_ACCESS constant is available for use with the InstallScript function **SetObjectPermissions**. You can pass this constant instead of the number 0 in the nOptions parameter of **SetObjectPermissions** to indicate that the permissions that are being set should allow access to the specified file, folder, or registry key.

This functionality is available in InstallScript events in the following project types: InstallScript and InstallScript MSI. This functionality is also available through InstallScript custom actions in Basic MSI, InstallScript MSI, and Merge Module projects.

For more information, see SetObjectPermissions

New Constants for the DialogSetInfo Function

Two new constants are available for the nParameter parameter of the InstallScript function **DialogSetInfo** when you are using DLG_INFO_ALTIMAGE for the nInfoType parameter:

- **DLG_INFO_ALTIMAGE_VERIFY_BMP**—This constant specifies that the bitmap that is indicated by szInfoString should be used in subsequent dialogs. Before this bitmap is used, the installation checks for the existence of the bitmap. If the bitmap does not exist, the function returns an error that indicates that the bitmap was not found.
- **DLG_INFO_ALTIMAGE_REVERT_IMAGE**—This constant specifies that dialogs should display the default bitmap. This constant is an alternative equivalent to the value of -1. The installation does not check for the existence of the bitmap when you use this constant.

If you pass TRUE in the nParameter parameter, the installation does not check for the existence of the bitmap; if the bitmap does not exist in this scenario, the function does not return an error.

This functionality is available in InstallScript events in the following project types: InstallScript and InstallScript MSI.

For more information, see DialogSetInfo.

Enhancements for .NET Framework 4.0 Support

A new FOLDER_DOTNET_40 InstallScript variable is available. This variable stores the path of the .NET Framework 4.0 files.

Two new constants are available for use with the function Is:

- REGDB_KEYPATH_DOTNET_40_CLIENT
- REGDB_KEYPATH_DOTNET_40_FULL

Path Variable Support for Each Patch Configuration's Output Location and Cache

The Common tab for a patch configuration that is selected in the Patch Design view contains two settings that let you browse to the appropriate folder: Patch Output Location and Patch Creation Cache. InstallShield now lets you use path variables for the folders instead of absolute paths. This support makes it easier to build releases on different build and development machines that use different directory structures.

This support is available if InstallShield is configured to use path variables on your system. (That is, the Path Variables tab on the Options dialog box must be configured to allow path variable support.) If you configure InstallShield to use absolute paths, the Patch Design view shows the absolute path to folders.

This enhancement is available in the following project types: Basic MSI and InstallScript MSI.

InstallScript Functions ServiceExistsService and ServiceGetServiceState No Longer Require Elevated Privileges

The InstallScript functions **ServiceExistsService** and **ServiceGetServiceState** no longer require elevated privileges. Therefore, installations can now call these functions when end users have limited privileges, such as in the Install UI sequence of a Basic MSI installation.

Important Information

Installing More than One Edition of InstallShield

Only one edition of InstallShield 2016—Premier, Professional, or Express—can be installed on a system at a time.

Installing More than One Version of InstallShield

InstallShield 2016 can coexist on the same machine with other versions of InstallShield.

The InstallShield 2016 Standalone Build can coexist on the same machine with other versions of the Standalone Build. In most cases, the Standalone Build is not installed on the same machine where InstallShield is installed. If you do install both on the same machine and you want to use the automation interface, review the Installing the Standalone Build and InstallShield on the Same Machine to learn about special registration and uninstallation considerations.

Integrating InstallShield with Visual Studio

Microsoft Visual Studio can be integrated with only one version of InstallShield at a time. The last version of InstallShield that is installed or repaired on a system is the one that is used for Visual Studio integration.

What's New in InstallShield 2010 Expansion Pack for Visual Studio 2010

InstallShield 2010 Expansion Pack for Visual Studio 2010 includes changes that offer support for the final released version of Visual Studio 2010 and .NET Framework 4. It also includes additional changes.

Microsoft .NET Framework 4 Prerequisites

InstallShield includes four new .NET-related InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Microsoft .NET Framework 4.0 Full
- Microsoft .NET Framework 4.0 Full (Web Download)
- Microsoft .NET Framework 4.0 Client
- Microsoft .NET Framework 4.0 Client (Web Download)

Note the following details about these prerequisites:

- The full prerequisites install the .NET Framework runtime and associated files that are required to run and develop applications that target the .NET Framework 4.
- The client prerequisites install the .NET Framework runtime and associated files that are required to run most client applications.

- The two Web download prerequisites require an Internet connection. These prerequisites download the required redistributable files if appropriate. The other two prerequisites are stand-alone installations that do not require an Internet connection.

The .NET Framework 4.0 requires Windows Installer 3.1 or later, as well as the Windows Imaging Component. Therefore, the .NET Framework 4.0 Full prerequisites are configured to have dependencies for the following InstallShield prerequisites:

- Windows Installer 3.1 (x86)
- Windows Imaging Component (x86)
- Windows Installer 3.1 for Windows Server 2003 SP1 (x86)
- Windows Imaging Component (x64)
- Windows Installer 3.1 for Windows Server 2003 SP1 (IA64)
- Windows Installer 3.1 for Windows Server 2003 SP1 (x64)
- Windows Installer 3.1 for Windows XP (x64)

In addition, the .NET Framework 4.0 Client prerequisites are configured to have dependencies for the following InstallShield prerequisites:

- Windows Installer 3.1 (x86)
- Windows Imaging Component (x86)
- Windows Installer 3.1 for Windows Server 2003 SP1 (x86)
- Windows Imaging Component (x64)
- Windows Installer 3.1 for Windows Server 2003 SP1 (x64)
- Windows Installer 3.1 for Windows XP (x64)

Thus, if you add any of the .NET Framework 4.0 prerequisites to your project, InstallShield also adds prerequisites for Windows Installer and Windows Imaging Component to your installation automatically by default. This could increase the size of your installation. In some cases, you may want to edit the .NET Framework 4 prerequisites in the InstallShield Prerequisite Editor (which is available in the Premier and Professional editions of InstallShield) to remove some of the dependencies. For example, if your product requires Windows Vista or later or Windows Server 2008 and later, you may want to remove the Windows Installer 3.1 dependencies that target earlier versions of Windows.

Additional Prerequisites for Visual Studio 2010 Support

InstallShield includes the following new InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Microsoft SQL CE 3.5 SP2
- Microsoft ReportViewer 2010
- Microsoft VSTO 2010 Runtime
- FSharp Redistributable Package 2.0
- Microsoft Office 2007 PIA (This prerequisite installs the Microsoft Office 2007 Primary Interop Assemblies. To use this prerequisite, download the **PrimaryInteropAssembly.exe** file from Microsoft's Web site and run it to

extract the **o2007pia.msi** file. Note that you may need to change the path of the **o2007pia.msi** installation in the .prq file, depending on where the .msi package is located on your system.)

Predefined System Searches for .NET Framework 4

InstallShield includes two new predefined system searches:

- Microsoft .NET Framework 4.0 Full package
- Microsoft .NET Framework 4.0 Client package

If your installation requires either of these, you can use the Installation Requirements page in the Project Assistant or the System Search view to add these system searches to your project. When end users launch your installation, Windows Installer checks the target system to see if the requirements are met; if they are not met, the installation displays the error message that is defined for the system search.

This functionality applies to Basic MSI and InstallScript MSI projects.

What's New in InstallShield 2010 SP1

InstallShield 2010 Service Pack 1 (SP1) includes changes that offer support for the final released versions of Windows 7, Windows Server 2008 R2, and Windows Installer 4.5. It also includes additional changes.



Important • *If you want to open an InstallShield 2010 project in InstallShield 2010 SP1, you must allow InstallShield to upgrade your project to InstallShield 2010 SP1. InstallShield 2010 SP1 includes support for tables that were not available in InstallShield 2010 projects, and these tables need to be added during the upgrade. Note that it is not possible to open InstallShield 2010 SP1 projects in earlier versions of InstallShield (including InstallShield 2010 without SP1). Therefore, if multiple users need to open and modify your InstallShield projects, ensure that all of you apply the SP1 patch at the same time.*

If you open an InstallShield 2010 project in InstallShield 2010 SP1, InstallShield 2010 SP1 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project before converting it.

Setup.exe Manifests Now Have Compatibility Section to Avoid Triggering Program Compatibility Assistant on Windows 7 and Windows Server 2008 R2 Systems

If you configure your InstallShield project to create a setup launcher for your installation, the manifest that InstallShield creates for the setup launcher now includes a compatibility section. Previously, without this compatibility section, a Program Compatibility Assistant (PCA) dialog box may have appeared at the end of an installation on Windows 7 and Windows Server 2008 R2 systems. The PCA dialog box indicated that the program might not have installed correctly. This dialog box was displayed if the installation did not create the application uninstallation key. This may happen if the end user cancels the installation or the installation fails to complete successfully.

Support for Creating App-V Package Upgrades and for Compressing App-V Packages

InstallShield now has support for creating App-V package upgrades. The Package Information page in the Microsoft App-V Assistant has a new Upgrade Settings link. Click this link to specify whether you want to create an upgrade. If you specify that you do want to create an upgrade, you can specify additional information about the upgrade, such as whether to append the version number to the App-V package file name. By default, App-V package upgrades are not created.

The Build Options page in the Microsoft App-V Assistant has a new setting that lets you specify whether you want to use compression for the data files in the App-V package. If you select Yes for this setting, InstallShield uses zlib compression for your App-V package.

The App-V upgrade and compression functionality is available in the following project types: Basic MSI and MSI Database.

The Microsoft App-V Assistant is available if you purchase the Premier Edition of InstallShield.

For more information, see:

- Specifying Upgrade Package Information
- Specifying Whether to Compress the Data Files in an App-V Package

Windows Installer 5 Support for Configuring New Customization Options for Windows Services, Plus Other Service-Related Improvements

InstallShield now includes support for configuring extended customization options for Windows services. The customization options include a new delayed auto-start capability to help system startup performance, improved failure detection and recovery options to improve system reliability, and more. Windows Installer 5 supports these new options; earlier versions of Windows Installer ignore them.

To configure the new service-related settings, use the new Services node in the Advanced Settings area for a component in the Setup Design view (in installation projects) or the Components view. All of the services for a component are now grouped and listed by service name under the Services node. In addition, the previously available service-related settings are now consolidated in the same grid with the new settings. In earlier versions of InstallShield, these previously available service-related settings were split among separate subviews for the Install NT Services and Control NT Services nodes. The consolidated grid of settings makes it easier to create a component that installs, configures, starts, stops, or deletes a new or existing service during installation or uninstallation.

This functionality is available in the following project types: Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

For more information, see:

- Installing, Controlling, and Configuring Windows Services
- Services Settings

Validation for the Windows 7 Logo Program

InstallShield includes two new validation suites: InstallShield Validation Suite for Windows 7 and InstallShield Merge Module Validation Suite for Windows 7. These validation suites can help you verify that your Windows Installer–based installation or merge module meets the installation requirements of the Compatible with Windows 7 logo program. If you want to be able to use the Windows 7 logo artwork, your product's installation must meet the program's requirements.

If you want to configure InstallShield to perform validation with these validation suites each time that a release is successfully built: On the Tools menu, click Options. On the Validation tab, select the appropriate check boxes.

If you want to perform validation separately from the build process: On the Build menu, point to Validation, and then click the appropriate new suite.

Windows Installer 5 Support for Setting Windows Shell Properties for a Shortcut

The Shortcuts view in InstallShield now lets you specify one or more shortcut properties that need to be set by the Windows Shell at run time. For example, if you do not want the Start menu entry for one of your shortcuts to be highlighted as newly installed after end users install your product, you can set a property for that shortcut. You might want to use this property with shortcuts that are for tools and secondary products that are part of your installation.

Windows Installer 5 includes support for setting Shell shortcut properties. Earlier versions of Windows Installer ignore those properties.

For more information, see Setting Shell Properties for a Shortcut.

Windows Installer 5 Support for the MsiPrint and MsiLaunchApp Control Events

When you are adding a control to a dialog in your project, you can select one of the new events that is now available for Windows Installer 5:

- **MsiPrint**—You can add this event to a push button control that is on a dialog that has a scrollable text control. When an end user clicks the push button control, the contents of the scrollable text control are printed.
- **MsiLaunchApp**—You can add this event to a check box control on a dialog, and select the file that you want to be launched in the event's Argument setting. The check box enables end users to choose whether to run the file at the end of the installation. This event is typically used with a check box control that is on the SetupCompleteSuccess dialog. The check box control should include a condition that prevents the control from being displayed during uninstallation.

Windows Installer 5 includes support for these control events. Earlier versions of Windows Installer ignore these events. Therefore, if your installation is run on a system that has Windows Installer 4.5 or earlier and you want to use one or both of these new events, add a condition to the dialog controls so that they are not displayed on systems that have Windows Installer 4.5 or earlier.

This functionality is available in Basic MSI projects.

Note that if you want to add the print or launch support to your project but your installation targets systems that have Windows Installer 4.5 or earlier, consider using the support that InstallShield provides. For information on adding the print support, see Specifying Dialogs for Your Installation in the Project Assistant. For information on adding the print support, see Adding a Print Button to a Dialog. The support that is available with InstallShield does not require Windows Installer 5.

Additional Changes

For a list of issues that are resolved in InstallShield 2010 SP1, see the release notes. The release notes are available from the Help menu in InstallShield.

What's New in InstallShield 2010

New Features

InstallShield includes the following new features.

Support for Creating Customized Virtual Applications

Now you can use InstallShield to create customized virtual applications in the Microsoft App-V format. Virtualization enables you to isolate an application in its own environment so that it does not conflict with existing applications or modify the underlying operating system.

Virtual applications run in virtual environments that keep the application layer and the operating system layer separate. Each application includes its own configuration information in its virtual environment. As a result, many applications can run side-by-side with other applications on the same computer without any conflicts.

To create a virtual application, use the new Microsoft App-V tab that is available in the following project types: Basic MSI and MSI Database.

Virtualization support is available if you purchase the Premier Edition of InstallShield.

InstallShield also includes InstallShield prerequisites for the Microsoft App-V 4.5 Desktop Client installation and the Microsoft Application Error Reporting installation. The Application Error Reporting prerequisite is a dependency of the App-V prerequisite. The redistributable files for these InstallShield prerequisites are not available for download from within InstallShield, since you must obtain them from Microsoft. Once you obtain them from Microsoft, place them in the location that is displayed when you are editing these prerequisites in the InstallShield Prerequisite Editor.

For more information, see:

- Creating Customized Virtual Applications
- About Virtualization
- Creating Microsoft App-V Applications

Ability to Target Windows 7 and Windows Server 2008 R2 Systems

InstallShield enables you to specify that your installation requires Windows 7 or Windows Server 2008 R2. It also lets you build feature and component conditions for these operating systems.

Windows 7 and Windows Server 2008 R2 Support for Displaying Installation Progress on the Taskbar

Installations that are run on Windows 7 and Windows Server 2008 R2 now show a progress bar on the Windows taskbar during file transfer. This applies to InstallScript and InstallScript MSI installations. In addition, it applies to Basic MSI installations that display billboards that were configured in the Billboards view. Note that a progress bar is not displayed on the taskbar on earlier versions of Windows. It is also not displayed during setup initialization or while InstallShield prerequisites are being installed.

Beta Windows Installer 5 Support for Per-User Installations

The General Information view has a new Show Per-User Option setting. This setting lets you specify whether you want the ReadyToInstall dialog—in certain scenarios—to include buttons that let end users indicate how they want to install the product: for the current user or for all users. The per-user button sets the new Windows Installer property **MSIINSTALLPERUSER** equal to 1 to indicate that the package should be installed for the current user. The **MSIINSTALLPERUSER** property is available with the beta of Windows Installer 5.

This feature is available in Basic MSI projects.

To learn more, see Per-User vs. Per-Machine Installations.

Beta Windows Installer 5 Support for Reducing the Time for Installing Large Packages

Use the new Fast Install setting in the General Information view to select one or more options that may help reduce the time that is required to install a large Windows Installer package. For example, you can specify that you do not want a system restore point to be saved for your installation. You can also specify that you want the installation to perform file costing, but not any other costing.

This setting configures the new Windows Installer property **MSIFASTINSTALL**, which can be set at the command line. Windows Installer 5 includes support for this property. Earlier versions of Windows Installer ignore it.

This setting is available in the following project types: Basic MSI, InstallScript MSI, MSI Database, and Transform.

To learn more, see the description of the Fast Install setting.

Beta Support for Additional Windows Installer 5 Features

The Director Editor in InstallShield includes beta support for the new Windows Installer 5 tables (**MsiLockPermissionEx**, **MsiServiceConfig**, **MsiServiceConfigFailureActions**, and **MsiShortcutProperty**).

This feature is available in the following project types: Basic MSI and InstallScript MSI.

Ability to Detect the Presence of a Virtual Machine

InstallShield lets you determine whether an installation is running on any of the following types of virtual machines:

- Microsoft Hyper-V
- A VMware product such as VMware Player, VMware Workstation, or VMware Server
- Microsoft Virtual PC

Two new Windows Installer properties are available when you add an MSI DLL custom action to your project to detect virtual machines: **IS_VM_DETECTED** and **IS_VM_TYPE**. The custom action should be configured to call the ISDetectVM function in the SetAllUsers.dll file, which is installed with InstallShield.

In addition, the InstallScript language has been expanded to support the detection. The structure SYSINFO contains a new bIsVirtualMachine member, and a new VIRTUAL_MACHINE_TYPE constant is available for use with the InstallScript function **GetSystemInfo**.

This feature is available in the following project types: Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module.

For more information, see Detecting Whether the Installation Is Being Run on a Virtual Machine.

Support for 64-Bit COM Extraction

InstallShield now supports 64-bit COM extraction. If you are using InstallShield on a 64-bit operating system, InstallShield can extract COM data from a 64-bit COM server. In order to install the data to the correct locations, the component must be marked as 64 bit.

InstallShield must be installed on a 64-bit operating system in order to perform 64-bit COM extraction.

This feature is available in the following project types: Basic MSI, InstallScript MSI, and Merge Module.

To learn more, see:

- Targeting 64-Bit Operating Systems
- Extracting COM Information from a COM Server

New Support for Setting Permissions for Files, Folders, and Registry Keys

InstallShield offers two new ways to secure files, folders, and registry keys for end users who run your product in a locked-down environment:

- **Custom InstallShield handling**—In Windows Installer–based projects, you can choose to use custom support for setting permissions at run time. With this option, InstallShield stores permission information for your product in the custom **ISLockPermissions** table of the .msi database. InstallShield also adds custom actions to your project to set the permissions. This support is available in the following project types: Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.
- **SetObjectPermissions, an InstallScript Function**—You can use the new **SetObjectPermissions** function in InstallScript events and InstallScript custom actions to set permissions at run time. You can use this function in the following project types: InstallScript, Basic MSI, InstallScript MSI, and Merge Module.

With the custom InstallShield handling option, the file, folder, or registry key for which you are setting permissions must be installed as part of your installation. With the **SetObjectPermissions** function, the file, folder, or registry key can be installed as part of your installation, or it can be already present on the target system.

Previously, the only option that InstallShield offered for setting permissions was to use the traditional Windows Installer handling. With this option, the permission information is stored in the **LockPermissions** table of the .msi database. The new custom InstallShield handling option and the **SetObjectPermissions** function offer several advantages over the traditional Windows Installer handling:

- The custom option and the **SetObjectPermissions** function include support for many well-known security identifiers (SIDs) that are not supported by the traditional Windows Installer handling option.
- The custom option and the **SetObjectPermissions** function support the use of localized user names for the supported SIDs, unlike the traditional option. With the traditional option, if you try to use a localized name to set permissions on a non-English system, the installation may fail.
- The custom option and the **SetObjectPermissions** function let you specify that you want to deny a user or group from having the permissions that you are specifying. The traditional handling does not allow you to do this.
- The custom option and the **SetObjectPermissions** function let you add permissions to a file, folder, or registry key that already exists on the target system, without deleting any existing permissions for that object. With the traditional handling, the existing permissions are deleted.
- The custom option and the **SetObjectPermissions** function let you configure permissions for a folder (or a registry key), and indicate whether you want the permissions to be applied to all of the folder's subfolders and

files (or the registry key's subkeys). With the traditional handling, if you want to configure permissions for a subfolder or a file in a folder (or a subkey under a registry key), the parent that is created on the target system automatically inherits the permissions of its child.

- The custom option and the **SetObjectPermissions** function let you configure permissions for a new user that is being created during the installation. The traditional handling does not allow you to do this; the user must already exist on the target system at run time.

The General Information view has a new Locked-Down Permissions setting that lets you specify whether you want to use the new custom InstallShield handling or the traditional Windows Installer handling for all new permissions that you set for files, folders, and registry keys in your project. If you have already configured some permissions in your project and then you change the value of this setting, InstallShield lets you specify whether you want to use the alternate handling method for those already-existing permissions. In all new projects, the default value for this setting is the custom InstallShield handling option. If you upgrade a project from InstallShield 2009 or earlier to InstallShield 2010, the traditional Windows Installer handling option is the default value of this setting. This new setting is available in the following project types: Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

For more information, see the following:

- Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment
- Selecting the Locked-Down Permissions Type for a Project
- SetObjectPermissions
- SetObjectPermissions Example

Support for InstallShield Prerequisites in InstallScript Projects

InstallShield now enables you to add InstallShield prerequisites to InstallScript projects. Previously, only Basic MSI and InstallScript MSI projects included support for this type of redistributable.

InstallShield prerequisites are redistributables that usually install a product or technology framework that is required by your product. Now you can add any of the InstallShield prerequisites that are provided with InstallShield—or any of your own custom-designed InstallShield prerequisites—to InstallScript projects. If you work on a mix of different project types, InstallShield lets you simplify your testing matrix by enabling you to reuse this type of redistributable in all of your Basic MSI, InstallScript, and InstallScript MSI projects.

To add an InstallShield prerequisite to an InstallScript project, use the new Prerequisites view. InstallScript projects include support for the setup prerequisite type of InstallShield prerequisite, which is run before the main installation's user interface is run. InstallScript projects do not have support for feature prerequisites, which are InstallShield prerequisites that are associated with features.

For more information, see:

- Adding InstallShield Prerequisites, Merge Modules, and Objects to InstallScript Projects
- Run-Time Behavior for an Installation that Includes InstallShield Prerequisites
- Specifying the Installation Order of InstallShield Prerequisites
- Specifying a Run-Time Location for a Specific InstallShield Prerequisite
- Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level
- Designing InstallShield Prerequisites and Other Redistributables

- Prerequisites View

New InstallShield Prerequisites for Windows Installer, .NET Framework, Crystal Reports, and Other Redistributables

InstallShield includes a number of new InstallShield prerequisites that you can add to Basic MSI, InstallScript, and InstallScript MSI projects:

- Windows Installer 4.5 (The InstallShield prerequisites for Windows Installer 4.5 include the fix that is described in Microsoft KB958655.)
- Windows Installer 4.5 Update (The InstallShield prerequisites for the Windows Installer 4.5 Update include the fix that is described in Microsoft KB958655. Windows Installer 4.5 must already be installed on the target system for this update.)
- Windows Installer 3.1, Windows Installer 3.0, and Windows Installer 2.0 (These versions of Windows Installer redistributables were previously available if you added Windows Installer to your project in the Releases view. These versions were not previously available as InstallShield prerequisites.)
- .NET Framework 3.0 SP1
- .NET Framework 2.0 SP2
- Internet Explorer 8
- Microsoft SQL Server 2008 Express SP1
- Microsoft SQL Server 2005 Express SP3
- Microsoft Visual C++ 2005 SP1 Redistributable Package
- Oracle 11g Instant Client 11.1.0.7 (Oracle does not provide an installer for the Oracle Instant Client files, so you need to create an .msi package before you can use this InstallShield prerequisite in your projects. You can do so easily by using the Oracle Instant Client installation project that is installed with InstallShield (*InstallShield Program Files Folder*\Support\Oracle Instant Client.)
- Crystal Reports Basic for Visual Studio 2008 (Use this prerequisite with the Crystal Reports Basic installation that is installed with Visual Studio 2008. Note that you may need to change the path of the Crystal Reports Basic installation in the .prq file, depending on where the .msi package is located on your system.)

Microsoft SQL Server 2008 SP1 Support

InstallShield now includes support for running SQL script on SQL Server 2008 SP1. In addition, InstallShield includes SQL Server 2008 SP1 in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports.

Oracle 11g Support

InstallShield now includes support for running SQL script on Oracle 11g. In addition, InstallShield includes Oracle 11g in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports.

New Tool for Scanning an IIS Web Site, Recording Its Settings, and Importing Those Settings into an InstallShield Project

InstallShield includes an IIS scanner (**IISscan.exe**), a new command-line tool for scanning an existing IIS Web site and recording IIS data about the Web site. The IIS scanner creates an XML file that contains all of the settings for the Web site, its virtual directories, its applications, and its application pools. You can use the XML file to import the IIS data into the Internet Information Services view in InstallShield. Once you have imported the IIS data into a project, you can use the Internet Information Services view to make changes to the IIS settings as needed.

The ability to import the IIS data into an InstallShield project is available only in the Premier edition of InstallShield.

For more information, see:

- Scanning an IIS Web Site and Importing Its Settings into an InstallShield Project
- Filtering IIS Data When Importing a Web Site and Its Settings into an InstallShield Project
- IISscan.exe

Ability to Add IIS Web Applications to Web Sites, Plus Other IIS-Related Improvements

InstallShield now lets you add IIS Web applications to Web sites. You can do so by right-clicking a Web site in the Internet Information Services view and clicking New Application. Once you have added a new application, you can configure its settings in the right pane.

InstallShield also lets you create a virtual directory without an application. Previously whenever you created a virtual directory, an application was also created automatically.

In addition, InstallShield has new IIS settings:

- **Managed Pipeline Mode**—This setting enables you to specify the appropriate request-processing pipeline mode—either integrated or classic—for an application pool.
- **Enable 32-Bit Applications**—This setting lets you specify whether you want to allow 32-bit applications in the selected application pool to be run on 64-bit systems.
- **.NET Framework Version**—This setting is where you specify the version of the .NET Framework that an application pool should load.
- **ASP.NET Platform**—If your installation may be run on a 64-bit version of Windows with the .NET Framework, use this setting to specify which ASP.NET platform should be used to map a Web site, application, or virtual directory to the ASP.NET version.

This feature is available in the following project types: Basic MSI, InstallScript, and InstallScript MSI.

To learn more, see:

- Creating a Web Site and Adding an Application or a Virtual Directory
- Internet Information Services View

New View for Configuring Run-Time Text File Changes

InstallShield has a new Text File Changes view, which enables you to configure search-and-replace behavior for content in text files—for example, .txt, .htm, .xml, .config, .ini, and .sql files—that you want to modify at run time on the target system. The text files can be part of your installation, or they can be files that are already present on target systems.

You can use Windows Installer properties to specify the names of the text files that you want to include in or exclude from your search. You can also use properties to specify the search strings and the replacement strings. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's text files are modified on the target system. For example, if your project includes a dialog in which end users must specify an IP address, your installation can search a set of files for a token, and replace it with the IP address that end users enter.

Note that the Text File Changes view offers an alternative for configuring XML file changes in the XML File Changes view. Using the Text File Changes view offers some advantages. For example, this new view does not have any run-time requirements; however, changes that are configured through the XML File Changes view require MSXML to be present on the target system. In addition, configuring changes in the Text File Changes view does not require you to enter XPath queries, as is required in the XML File Changes view.

The Text File Changes view is available in the following project types: Basic MSI and InstallScript MSI.

For more information, see:

- Modifying Text Files
- Creating a Text Change Set
- Specifying a Change for a Text File
- Changing the Order in Which Text File Changes Are Made
- Using Windows Installer Properties to Dynamically Modify Text Files
- Specifying the Code Page that Should Be Used for Opening ANSI Text Files

New String Editor View

InstallShield has a new String Editor view. This view contains a spreadsheetlike table that shows the collection of language-independent identifiers and corresponding language-specific values for your project. In the String Editor view, you have complete and centralized control over the localizable text strings that are displayed at run time during the installation process. The view replaces the string tables that were previously available as nodes within the General Information view. Following is a list of some of the highlights of the new view:

- The view has a toolbar with buttons that let you add, edit, delete, find, replace, export, and import string entries. It also has a button that lets you search the project to identify all of the instances in which a specific string identifier is used.
- The top of the view has a new group box area; you can drag and drop column headings onto this area to organize the rows in the view in a hierarchical format. This functionality makes it easy to sort string entries by categories such as language and by modified date.
- This view has support for dynamic searches—as you are typing a string in the search box, InstallShield hides all of the string entries that do not contain it.

This feature is available in the following project types: Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module.

For more information, see:

- String Editor View
- Localizing the End-User Interface

Support for Unicode

InstallShield takes a three-pronged approach to fully supporting modern multilanguage installations: Windows Installer databases can now be built in a Unicode format, InstallShield projects are now stored in a Unicode format, and the InstallShield interface now supports entering and viewing Unicode characters from multiple character sets at the same time.

Unicode (UTF-8) Databases

The Build tab in the Releases view has a new Build UTF-8 Database setting that lets you specify that a Windows Installer database, along with any instance or language transforms, be built using the UTF-8 encoding. The UTF-8 encoding supports characters from all languages simultaneously, enabling you to mix and match, for example, Japanese and German, or Russian and Polish, both in text shown to end users and in file names and registry keys. These mixed languages work correctly regardless of the current language of the target system. Unicode support has also been added to key parts of the installation run times, including IIS support and changes to text and XML files.

The default value for the new Build UTF-8 Database setting is No.

The automation interface now includes support for this new setting. The ISWiRelease object includes a new BuildUTF8Database property that lets you specify whether you want to use the UTF-8 encoding.

This feature is available in the following project types: Basic MSI, InstallScript MSI, and Merge Module projects.

To learn more, see the description of the Build UTF-8 Database setting.

Unicode Project Files (*.ism)

InstallShield now uses the UTF-8 encoding when saving both binary and XML project files. Because of this change, the files, registry data, and other strings that are used in the project can include characters from all languages simultaneously. With this encoding, InstallShield no longer needs to use an unreadable Base64 encoding for strings that are stored in the **ISString** table. Instead, as you add or modify strings in a project, InstallShield now saves them as human-readable Unicode strings that you can easily examine for changes across revisions of your project. Therefore, InstallShield uses only Unicode strings for all new projects that are created in InstallShield 2010; for upgraded projects, InstallShield uses Unicode for new and modified strings, as well as for strings that have been exported and reimported.

If you use Unicode values that cannot be represented in the target build (for example, an InstallScript installation, or a Basic MSI installation in which No is selected for the Build UTF-8 Database setting), a new build error points to the item that needs to be changed. In some instances, this reveals invalid string entries that were silently allowed in earlier versions of InstallShield.

This feature applies to all project types.

Unicode Views in InstallShield

Many views in InstallShield have been enhanced to display and allow you to enter characters from all languages. For example, in the Files and Folders view, Chinese file names from your local English system are no longer displayed with question marks for their names, and now you can add these files to your project. Similarly, the Registry view no longer converts Thai registry keys or values to question marks, and you can install them with your Windows Installer–based projects. In addition, you can view and edit strings from all languages in the String Editor view, a new separate view; previously, string entries were available from separate language nodes in the General Information view. Examples of other enhanced views include the Property Manager, Path Variables, Direct Editor, General Information, and Setup Design views.

Note that whenever No is selected for the Build UTF-8 Database setting, all file names, registry keys, and other strings must still consist of characters from the code page of all target languages that will use it. In this scenario, if an item uses a character that is not available on the code page of a target language, InstallShield reports a new build error that points to this item; the Chinese file name cannot be used in an English installation unless Yes is selected for the Build UTF-8 Database setting.

This feature applies to all project types.

New Billboard Support in Basic MSI Projects

Basic MSI projects now include support for billboards. You can add billboards to your projects to display information to end users during the installation process. The billboards can be used to communicate, advertise, educate, and entertain end users. For example, billboards can present overviews on new features of the product being installed or other products from your company. Each billboard is a file that you or your company's graphics department creates for complete control over the look and feel of the file transfer.

Following are some of the highlights regarding billboard support in Basic MSI projects:

- You can add an Adobe Flash application file (.swf) as a billboard in your project. Flash application files can consist of videos, movies, sounds, interactive interfaces, games, text, and more—anything that is supported by the .swf type of file.
- InstallShield lets you use .bmp, .gif, .jpg, and .jpeg files as billboards.
- InstallShield includes a Billboard Type setting that lets you specify which style of billboard you want to use in your installation. For example, with one style, the installation displays a full-screen background, with billboards in the foreground, and a small progress box in the lower-right corner of the screen. With another style, the installation displays a standard-size dialog that shows the billboards. The bottom of this dialog shows the progress bar.
- InstallShield lets you preview a billboard to see how it would be displayed at run time, without requiring you to build and run a release. Previewing a billboard lets you see how your billboard will look with the background color, position, and related settings that are currently configured for your billboard.

The Billboards view in InstallShield is where you add billboard files, configure billboard-related settings, and preview billboards.

To learn more, see:

- Displaying Billboards in Basic MSI Installations
- Billboard File Types for Basic MSI Projects
- Types of Billboards for Basic MSI Projects
- Adding an Adobe Flash Application File Billboard to a Basic MSI Project
- Adding Image Billboards to a Basic MSI Project
- Configuring Billboard Settings in a Basic MSI Project
- Previewing Billboards Without Building and Launching a Release
- Setting the Billboard Order in a Basic MSI Project
- Run-Time Behavior of a Basic MSI Installation that Includes Billboards

Previously, only InstallScript and InstallScript MSI projects included support for billboards. Note that billboard support in these project types is different than support in Basic MSI projects. For more information, see Displaying Billboards.

Expanded Billboard Support in InstallScript and InstallScript MSI Projects

InstallScript and InstallScript MSI projects now let you use .gif, .jpg, and .jpeg files as billboards. Previously, only .bmp files were supported. Use the Support Files/Billboards view to add billboards to your project.

Billboard files in InstallScript and InstallScript MSI projects must follow a designated naming convention. Each file name must begin with **bbrd**, followed by the number of the billboard (from 1 through 99); each must end with a supported file extension (.bmp, .gif, .jpg, or .jpeg). Note that it is no longer necessary for the file name numbers to be contiguous; that is, you can add **bbrd1.jpg**, **bbrd3.jpg**, and **bbrd5.jpg** to your project, and each image is displayed at run time in order. Previously, there could not be any numbers missing from the file name sequence for your billboards. For example, if you added **bbrd1.bmp**, **bbrd2.bmp**, and **bbrd4.bmp** to your installation project, then **bbrd1.bmp** and **bbrd2.bmp** were displayed at run time; however, **bbrd4.bmp** was not displayed, since **bbrd3.bmp** was missing from the sequence.

For more information, see:

- Displaying Billboards in InstallScript and InstallScript MSI Installations
- Naming Billboard Files in an InstallScript or InstallScript MSI Project
- Adding an Image Billboard to an InstallScript or InstallScript MSI Project

Ability to Play an Adobe Flash Application File (.swf) with the InstallScript Function PlayMMedia

The InstallScript function **PlayMMedia** now includes support for playing an Adobe Flash application file (.swf) on a background window during InstallScript and InstallScript MSI installations. Flash application files can consist of videos, movies, sounds, interactive interfaces, games, text, and more—anything that is supported by the .swf type of file.

If you are using a Flash file, you can use **SizeWindow** and **PlaceWindow** to control the size and placement of the background window that displays the Flash file.

For more information, see:

- PlayMMedia
- SizeWindow
- PlaceWindow
- Displaying a Background Window in InstallScript and InstallScript MSI Installations

Support for HTML Controls on Dialogs During InstallScript and InstallScript MSI Installations

InstallShield includes support for HTML controls on dialogs in InstallScript and InstallScript MSI projects. HTML controls enable you to use HTML markup for dialog controls. You can include on dialogs links to Web pages, installed HTML files, and HTML support files. If an end user clicks the hyperlink on the run-time dialog, you can have the HTML page open in an Internet browser, or you can trigger other behavior that you have defined through your InstallScript code. The HTML control lets you use any valid HTML markup, including styles to control their appearance.

The HTML control also lets you display the HTML content directly on a dialog if the content is an installed HTML file or HTML support file.

A new InstallScript function called **CtrlGetUrlForLinkClicked** is available. This function retrieves the URL for a link that an end user clicked.

For more information, see:

- Using an HTML Control on a Dialog
- CtrlGetUrlForLinkClicked
- CtrlGetUrlForLinkClicked Example

Ability to Add Unsupported Languages to InstallScript Projects

The InstallShield Premier edition includes default run-time strings for InstallScript projects in 33 different languages. The InstallShield Premier edition also lets you use the New Language Wizard to add unsupported languages, beyond the built-in 33 languages, to InstallScript projects. Once you have added unsupported languages to an InstallScript project, you can use the String Editor view to provide translated strings for the unsupported languages. As an alternative, you can export a language's string entries to a file, translate the string values in the file, and then import the translated file into your project. Previously, the New Language Wizard enabled you to add unsupported languages to only Basic MSI and InstallScript MSI projects.

For more information, see:

- Run-Time Language Support in InstallShield
- New Language Wizard
- String Editor View

Ability to Specify Paths for Libraries to Which the InstallScript Compiler Should Link

The Compile/Link tab on the Settings dialog box has a new Additional Library Paths box that lets you specify the locations where the InstallScript compiler should search for InstallScript libraries (.obl files) that should be linked to your script. In addition, when you specify your custom libraries on this tab, you can now specify just the file name, without the full path. These changes enable you to move your libraries to different directories but still successfully compile your script without manually changing the path for libraries on the Compile/Link tab.

For more information, see Compile/Link Tab.

New FlexNet Connect 11.6 Redistributables Available

InstallShield includes support for FlexNet Connect 11.6 in Basic MSI and InstallScript MSI projects. Use the Update Notifications view in InstallShield to include one of the two FlexNet Connect 11.6 merge modules—one has the Common Software Manager, and the other does not. These merge modules replace the FlexNet Connect 11 merge modules.

DIFx 2.1.1 Support

InstallShield includes support for the latest version of Driver Install Frameworks for Applications (DIFx 2.1.1). This new version, which includes the latest binary files from Microsoft, is available for the following project types: Basic MSI, InstallScript, and InstallScript MSI.

Enhancements

InstallShield includes the following enhancements.

Usability Enhancements

Many of the views in InstallShield have been enhanced to improve productivity and usability. For example, several views contain new toolbars that make options easier to find. Some of the views that contain grids let you customize how the rows in a grid are organized. Searches are performed more quickly in the views that offer search capabilities. Following are examples of some of the highlights:

- Direct Editor view—When you select a table in this view, a new toolbar is displayed. The toolbar has buttons that let you add, cut, copy, paste, find, and replace data in the table. This view also has support for dynamic searches—as you are typing a string in the search box, InstallShield hides all of the table records that do not contain it. When you are adding or editing a record, InstallShield displays help text for each field.
- Property Manager view—This view contains a new toolbar that has buttons for performing tasks such as adding and deleting properties. This view also has support for dynamic searches—as you are typing a string in the search box, InstallShield hides all of the table records that do not contain it. The top of the view has a new group box area; you can drag and drop column headings onto this area to organize the rows in the view in a hierarchical format. You can now select multiple properties in this view (by using the mouse and the SHIFT or CTRL button) and then delete them all at once.
- Redistributables view—The new toolbar and the new group box area in this view provide robust search and organizational functionality. You can drag and drop column headings onto the group box area to organize the list of redistributables in a hierarchical format. In addition, you can type a string in the toolbar's search box, and InstallShield hides all of the redistributables that do not contain it.
- Internet Information Services view—This view has been redesigned to look similar to IIS 7: the settings are now displayed in grids, instead of on tabs. The grids have buttons that let you sort the grid settings by category or alphabetically. When you select a setting in one of the grids in this view, InstallShield displays help information for that setting in the lower-right pane.
- General Information view—All of the settings in this view are displayed in one grid, instead of as separate grids that are associated with nodes within this view. The settings are grouped into several categories to make it easy to find a particular setting. In addition, the grid has a button that lets you sort the grid settings by category or alphabetically. The string tables that were previously in this view have been moved to a new String Editor view.
- Path Variables view—This view contains a new toolbar that has buttons for performing tasks such as adding and deleting path variables. This view also has support for dynamic searches—as you are typing a string in the search box, InstallShield hides all of the rows that do not contain it. The top of the view has a new group box area; you can drag and drop column headings onto this area to organize the rows in the view in a hierarchical format. You can now select multiple path variables in this view (by using the mouse and the SHIFT or CTRL button) and then delete them all at once.

In addition, the Output window, which is displayed when you are building a release, performing validation, or compiling script, has been enhanced. The Output window or its individual tabs can be docked to any side of the workspace in InstallShield, or they can be dragged to free-floating positions. If you drag the Output window or one of its tabs to the edge of the InstallShield interface, it becomes a docked window. If you drag the Output window or one of its tabs away from any of the edges of the InstallShield interface, it becomes undocked.

For more information, see the following:

- Working with the Group Box Area in Various Views
- Docking or Undocking the Output Window

Support for Specifying Action Progress Messages

To keep end users informed, installations commonly display text on the progress dialog to describe the installation's current activity. This usually accompanies the progress bar as a means of installation status. As each standard action and custom action is encountered, a message about the action is displayed on the progress dialog. This may be especially useful for actions that take a long time to execute. The same action text is also written to the installation's log file if one is created.

The Custom Actions and Sequences view has a new Action Text area that lets you specify action descriptions and details. This is available in the following project types: Basic MSI, InstallScript MSI, MSI Database, and Transform.

For more information, see:

- Using Action Text
- Specifying an Action Description and a Template for Action Data
- Displaying Action Descriptions on the Progress Dialog
- Displaying Action Data on the Progress Dialog

Ability to Detect Whether a 64-Bit System Allows 32-Bit IIS Applications to Be Run

At run time, you may need to have your installation check the Enable32bitAppOnWin64 property on systems that have IIS 6. Depending on the requirements for your product and the results of that check, you may want the installation to skip a particular component that contains a 32-bit IIS 6 application, or a 64-bit IIS application, for example, and proceed with the rest of the file transfer.

InstallShield includes a sample Windows Installer DLL file that detects how the Enable32bitAppOnWin64 property is set on a target system. You can add a custom action for this DLL to your project. If 32-bit applications are allowed, the Windows Installer property **ISIIS6APPPOOLSUPPORTS32BIT** is set to a value of 1; if they are not allowed, this property is not set. You can use this property in conditions to prevent or trigger certain behavior. For example, you can create a launch condition if you want the installation to exit if the **ISIIS6APPPOOLSUPPORTS32BIT** is set or not set.

This feature is available in the following project types: Basic MSI and InstallScript MSI.

For instructions on how to add the custom action to your project, see Considerations for Supporting IIS 6 on 64-Bit Platforms.

Ability to Detect Whether the IIS 6 Metabase Compatibility Feature Is Installed

At run time, you can have your installation detect if the IIS Metabase and IIS 6 Configuration Compatibility feature is installed on a target system, or if IIS 6 or earlier is installed. Depending on the requirements for your product and the results of that detection, you may want the installation to exit and display an error message.

For example, Web service extensions can be installed on systems that have IIS 6. On systems that have IIS 7, Web service extensions can be installed only if the IIS Metabase and IIS 6 Configuration Compatibility feature is installed. Thus, you might want to configure your installation to verify that IIS 6 is present or the IIS 6 compatibility feature is installed; if either of those conditions are false, the installation would exit and display an error message.

InstallShield includes a sample Windows Installer DLL file that detects whether the IIS Metabase and IIS 6 Configuration Compatibility feature is installed. You can add a custom action for this DLL to your project. If the IIS Metabase and IIS 6 Configuration Compatibility feature is installed, the Windows Installer property

ISIISMETABASECOMPATPRESENT is set to a value of 1; if it is not installed, this property is not set. You can use this property in conditions to prevent or trigger certain behavior. For example, you can create a launch condition if you want the installation to exit if the **ISIISMETABASECOMPATPRESENT** is set or not set.

This feature is available in the following project types: Basic MSI and InstallScript MSI.

For instructions on how to add the custom action to your project, see Determining If a Target System Has IIS 6 or Earlier or the IIS 6 Metabase Compatibility Feature.

Ability to Associate a Namespace Prefix with an Attribute of an XML File's Element

InstallShield now lets you add a namespace prefix to an attribute in the XML File Changes view. To do so, you can type the prefix, followed by a colon (:), in front of the attribute name. Previously, if an attribute contained a namespace prefix, the installation failed.

This enhancement applies to the following project types: Basic MSI, InstallScript, and InstallScript MSI.

For more information, see Adding a Namespace Prefix to an Attribute.

Ability to Specify Whether an XML File Should Be Formatted After Run-Time Changes

The Advanced tab for an XML file in the XML File Changes view has a new **Format XML after changes** check box that lets you specify whether you want the XML file to be formatted after the run-time changes are made to the file. When formatting the file, the installation adds indentations to the file and replaces empty-element tags with start tags and end tags. This may cause problems for **web.config** files, so you may want to clear this check box for your project.

This enhancement applies to the following project types: Basic MSI, InstallScript, and InstallScript MSI.

For more information, see Advanced Tab for an XML File.

Predefined System Searches for the .NET Framework and Internet Explorer 8

InstallShield has two new predefined system searches:

- Microsoft .NET Framework 3.5 SP1
- Internet Explorer 8

If your installation requires any of these, you can use the System Search view or the Installation Requirements page in the Project Assistant to add these system searches to your project. When end users launch your installation, Windows Installer checks the target system to see if the requirements are met; if they are not met, the installation displays the error message that is defined for the system search.

This enhancement applies to Basic MSI and InstallScript MSI projects.

Custom Properties Are Now Listed in Condition Builder

The Properties list on the Condition Builder dialog box now includes properties that are added in the Property Manager view. It also includes properties from searches that are created through the System Search Wizard. Therefore, when you are creating or editing a condition in InstallShield, you can now select your own custom properties without having to manually type them.

This enhancement applies to the following project types: Basic MSI, InstallScript MSI, Merge Module, and Transform.

Ability to Specify a Maximum Service Pack Number and 64-Bit Locations for InstallShield Prerequisite Conditions

The Prerequisite Condition dialog box, which is displayed when you are adding or modifying a condition for an InstallShield prerequisite in the InstallShield Prerequisite Editor, has a new field that lets you specify the maximum service pack number. Previously, it was possible to specify a minimum service pack number, but not a maximum service pack number.

In addition, the Prerequisite Condition dialog box now includes the following 64-bit locations in the box that is displayed when you are specifying file-related conditions: [CommonFiles64Folder], [ProgramFiles64Folder], and [System64Folder]. You can select any of these properties if you want to use 64-bit locations in the file path. At run time, the installation checks these 64-bit locations if the target system is 64 bit. For 32-bit systems, the installation checks the 32-bit location equivalents.

For more information, see Prerequisite Condition Dialog Box.

Enhancements to MSBuild Support

The InstallShield task for MSBuild includes several new parameters:

- **RunMsiValidator**—Use this parameter to specify the .cub file that you want to use for validation. This parameter is exposed as the ItemGroup InstallShieldMsiValidators when the default targets file is used.
- **PatchConfiguration**—Use this parameter to specify the patch configuration that you want to build through MSBuild. This parameter is exposed as the property InstallShieldPatchConfiguration when the default targets file is used.
- **PathVariables**—Use this parameter to override the value of a path variable. This parameter is exposed as the ItemGroup InstallShieldPathVariableOverrides when the default targets file is used.
- **PreprocessorDefines**—Use this parameter to specify preprocessor definitions for compiling InstallScript. This parameter is exposed as the ItemGroup InstallShieldPreprocessorDefines when the default targets file is used.

For more information, see Microsoft Build Engine (MSBuild).

Automation Interface Enhancements

OSFilter Property Values for Windows 7, Windows Server 2008 R2, Windows Vista, Windows Server 2008, and Windows Server 2003

The following constants are now available for use with the OSFilter member of the ISWiComponent and ISWiRelease objects in the automation interface:

- eosWin7 (33554432)—This is for Windows 7 and Windows Server 2008 R2.
- eosWinVista (16777216)—This is for Windows Vista and Windows Server 2008.
- eosWinServer2003 (8388608)

In addition, the value for the eosAll constant is now 64028880; previously, it was 5308624.

The OSFilter member applies to the ISWiComponent object in InstallScript, InstallScript MSI, and InstallScript Object projects. The OSFilter member applies to the ISWiRelease object in InstallScript and InstallScript Object projects.

For more information, see the following:

- ISWiComponent Object

- ISWiRelease Object

New IsPlatformSelected Property Values

The list of available property values for the IsPlatformSelected property of the ISWiComponent object has been expanded. This property now has values for 32-bit, 64-bit Itanium, AMD64, and Windows Server 2003 R2. This applies to the following project types: InstallScript and InstallScript Object.

Note that some of the values for the existing constants have changed. To learn the new values, see ISWiComponent Object.

New AddSQLScriptEx Method for Adding a SQL Script to a Connection

The AddSQLScriptEx method has been added to the ISWiSQLConnection object. Use this method to add an ISSQLScriptFile entry and generate a valid name from the passed string. The method ensures that the names of the entries in the ISSQLScriptFile table are unique and less than 47 characters in length.

New RunOnLogon and Condition Properties for the ISWiSQLScript Object

The read-write RunOnLogon property has been added to the ISWiSQLScript object. This property corresponds with the Run Script During Login check box on the Runtime tab for a SQL script in the SQL Scripts view.

The read-write Condition property has also been added to the ISWiSQLScript object. This property specifies the condition that is evaluated at run time to determine whether the SQL script should be run during installation or uninstallation. If the condition evaluates to true, the script is run. This property is available for the following project types: Basic MSI and InstallScript MSI.

New DisplayName Property for the ISWiUpgradeTableEntry Object

The read-write DisplayName property has been added to the ISWiUpgradeTableEntry object. This property gets or sets the name of an upgrade entry. This is the internal name that is displayed for an upgrade item in the Upgrades view. This property is available for the following project types: Basic MSI and InstallScript MSI.

Enhancements to the InstallScript Language for Operating Systems

The following structure members and predefined constants were added to the InstallScript language:

- **SYSINFO.WINNT.bWin7_Server2008R2**—This is a new SYSINFO structure member. If the operating system is Windows 7 or Windows Server 2008 R2, this value is TRUE.
- **SYSINFO.bWinServer2003R2**—This is a new SYSINFO structure member. If the operating system is Windows Server 2003 R2, this value is TRUE. (Note that the value of SYSINFO.WINNT.bWinServer2003 is also TRUE on this operating system.)
- **ISOSL_WIN7_SERVER2008R2**—This is a new predefined constant that is available for use with the **FeatureFilterOS** function and the SYSINFO structure variable. It indicates that the target system is running Windows 7 or Windows Server 2008 R2.
- **ISOS_ST_SERVER2003_R2**—This is a new operating system suite that is available for use with the **FeatureFilterOS** function and the SYSINFO structure variable. It indicates that the target system is running Windows Server 2003 R2.

In addition, some items were renamed:

- The SYSINFO.WINNT.bWinVista structure member does not distinguish between Windows Vista or Windows Server 2008. Therefore, the new member SYSINFO.WINNT.bWinVista_Server2008 is now available. The old alias is still available, but the new one may be preferred for clarity in code.
- The ISOSL_WINVISTA predefined constant does not distinguish between Windows Vista or Windows Server 2008. Therefore, the new constant ISOSL_WINVISTA_SERVER2008 is now available. The old alias is still available, but the new one may be preferred for clarity in code.

For more information, see FeatureFilterOS function and the SYSINFO structure variable.

Ability to Easily Override InstallScript Dialog Source Code in the InstallScript View

The event category drop-down list in the InstallScript view has a new Dialog Source option. If you select this option, the event handler drop-down list shows all of the built-in InstallScript dialogs. You can select any dialog in this list to modify its code.

This functionality applies to the following project types: InstallScript, InstallScript MSI, and InstallScript Object.

For information about built-in InstallScript dialog functions that are available when you select the Dialog Source option, see Dialog Functions.

Changes to the Major and Minor Version Registry Entries for the Uninstall Key of InstallScript Installations

InstallScript installations now create VersionMajor and VersionMinor registry values in the Uninstall key; the names of these values now match the entries that are created during Basic MSI and InstallScript MSI installations. This applies to new installations that are created in InstallShield 2010, as well as installations that are upgraded from InstallShield 2009 or earlier. Previously, in InstallShield 2009 and earlier, the names of the values that InstallScript installations created were MajorVersion and MinorVersion; these are no longer created.

In order to use the new registry values, the values of the following InstallScript constants have been changed:

- REGDB_VALUENAME_UNINSTALL_MAJORVERSION is now VersionMajor instead of MajorVersion.
- REGDB_VALUENAME_UNINSTALL_MINORVERSION is now VersionMinor instead of MinorVersion.

When the **MaintenanceStart** function is called, it creates the updated value names in the registry. By default, it also deletes the old value names if they exist. If you do not want the old value names to be deleted from target systems, you can use the new REGDB_OPTIONS option called REGDB_OPTION_NO_DELETE_OLD_MAJMIN_VERSION.

If REGDB_UNINSTALL_MAJOR_VERSION or REGDB_UNINSTALL_MINOR_VERSION is used with the **RegDBGetItem** function, **RegDBGetItem** first checks for the new value; if the new value is found, the function returns the value data from the new value. If the new value is not found, the function automatically checks for the old value; if the old value is found, the function returns the value data from the old value.

To provide backwards compatibility, the following new constants are available:

- REGDB_UNINSTALL_MAJOR_VERSION_OLD
- REGDB_UNINSTALL_MINOR_VERSION_OLD

You can specify these constants with the **RegDBGetItem**, **RegDBSetItem**, and **RegDBDeleteItem** functions to get, set, and delete the old values.

The following new string constants are also available:

- REGDB_VALUENAME_UNINSTALL_MAJORVERSION_OLD is defined as MajorVersion.

- REGDB_VALUENAME_UNINSTALL_MINORVERSION_OLD is defined as MinorVersion.

For more information, see the following:

- REGDB_VALUENAME_UNINSTALL_MAJORVERSION
- REGDB_VALUENAME_UNINSTALL_MAJORVERSION_OLD
- REGDB_VALUENAME_UNINSTALL_MINORVERSION
- REGDB_VALUENAME_UNINSTALL_MINORVERSION_OLD
- REGDB_UNINSTALL_MAJOR_VERSION_OLD
- REGDB_UNINSTALL_MINOR_VERSION_OLD
- REGDB_OPTIONS
- RegDBSetItem
- RegDBGetItem
- RegDBDeleteItem

New CtrlGetDlgItem Function for Retrieving the Window Handle of a Control in a Custom InstallScript Dialog

A new InstallScript function called **CtrlGetDlgItem** is now available. The **CtrlGetDlgItem** function retrieves the window handle of a control in a custom dialog. **CtrlGetDlgItem** is similar to the Windows API **GetDlgItem**, except that with **CtrlGetDlgItem**, you can specify the InstallScript dialog name instead of the dialog's window handle. For more information, see CtrlGetDlgItem.

New InstallScript Constant for Passing a Null Pointer for an InstallScript String to an External DLL Function

You can use the IS_NULLSTR_PTR variable to pass a null pointer to an external DLL function or Windows API through a parameter that has been prototyped as an InstallScript string. This functionality works for byval string, byref string, wstring, and binary data types. For more information, see IS_NULLSTR_PTR.

New StrConvertSizeUnit Function for Converting an InstallScript Size Unit Constant to a Display String

A new InstallScript function called **StrConvertSizeUnit** is now available. The **StrConvertSizeUnit** function returns the appropriate display string for the InstallScript size unit constant that is specified.For more information, see StrConvertSizeUnit.

New StrTrim Function for Removing Leading and Trailing Spaces and Tabs from a String

A new InstallScript function called **StrTrim** is now available. The **StrTrim** function removes the leading and trailing spaces and tabs from a string. For more information, see StrTrim.

New SdLicense* Dialog Functions to Supersede Existing SdLicense* Dialog Functions

Two new InstallScript dialog functions—**SdLicenseEx** and **SdLicense2Ex**—are now available. They both display a dialog that contains a license agreement in a multi-line edit field. The license agreement can be stored in a text file (.txt) or a rich text file (.rtf).

- SdLicenseEx displays a dialog that shows a question in a static text field. The end user responds by clicking the Yes or No button.

SdLicenseEx supersedes **SdLicense** and **SdLicenseRtf**.

- SdLicense2Ex displays a dialog that has two radio buttons (one for accepting the terms of the license agreement, and one for not accepting them). The Next button becomes enabled when the end user clicks the appropriate button to accept the terms of the license agreement.

  SdLicense2Ex supersedes **SdLicense2** and **SdLicense2Rtf**.

New ListFindKeyValueString Function for Searching Lists of Key-Value Pairs

A new InstallScript function called **ListFindKeyValueString** is now available. The **ListFindKeyValueString** function searches a string or number list for a specified value. It returns a value from an additional list that corresponds with the position of the found string in the first list. This enables you to search lists of key-value pairs for a particular key and retrieve the corresponding value.

For more information, see ListFindKeyValueString.

New InstallScript Code Examples

The InstallShield documentation now has sample code for the following InstallScript functions:

- AdminAskPath
- CharReplace
- FormatMessage
- LogReadCustomNumber
- LogReadCustomString
- LogWriteCustomNumber
- LogWriteCustomString

You can copy this code from the InstallShield documentation, paste it into your InstallScript code, and customize it as necessary.

Expanded InstallScript Cabinet File Viewer

The Cabinet File Viewer now provides additional information about .cab files for InstallScript projects.

- For features that were built by InstallShield, the viewer has the following new fields: Password Protected, Split Before, Split After, Split Not Allowed, and Split Before Not Allowed, and Image Index.
- For InstallScript Objects that are included in the InstallScript installation, the viewer has a new Object version field.
- For components, the viewer has the following new fields: Encrypted, Data as Files, and .NET Assembly.
- For media, the viewer has the following new field: Executable File.

What's New in InstallShield 2009 SP2

InstallShield 2009 Service Pack 2 (SP2) includes the following changes:

Microsoft Visual Studio 2008 SP1 Support

InstallShield now supports Microsoft .NET Framework 3.5 SP1 and Microsoft SQL Server 2008, as described below.

In addition, InstallShield integration with Visual Studio supports Visual Studio 2008 SP1, enabling development of installations and products within the Visual Studio interface.

Microsoft .NET Framework 3.5 SP1 Prerequisite Available

InstallShield includes two new .NET-related InstallShield prerequisites that you can add to Basic MSI and InstallScript MSI projects:

- Microsoft .NET Framework 3.5 SP1 (Web Download)
- Microsoft .NET Framework 3.5 SP1 (Full Package)

To learn more, see Adding .NET Framework Redistributables to Projects.

Microsoft SQL Server 2008 Support

InstallShield now includes support for running SQL script on SQL Server 2008. In addition, InstallShield includes SQL Server 2008 in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports.

Microsoft SQL Server 2008 Express Prerequisites Available

InstallShield now includes InstallShield prerequisites for Microsoft SQL Server 2008 Express Edition. You can add these InstallShield prerequisites to Basic MSI, and InstallScript MSI projects.

In order to support these new prerequisites, the InstallShield Prerequisite Editor has a new registry condition type.

New Registry Condition Type in the InstallShield Prerequisite Editor

The InstallShield Prerequisite Editor has a new type of condition option called **A registry entry has a specified version value**. Use this condition type when you want the installation to check the target system to see if a version number that is stored as the value data of a particular registry entry is greater than, less than, or equal to a version number that you specify in the InstallShield Prerequisite Editor. For more information about this new condition, see Prerequisite Condition Dialog Box.

Microsoft SQL Server 2005 Express SP2 (x86 and x64 WOW) Prerequisite Available

InstallShield now includes a new InstallShield prerequisite for Microsoft SQL Server 2005 Express Edition SP2. This prerequisite installs Microsoft SQL Server 2005 Express Edition SP2 on 32-bit and 64-bit (WOW) systems. You can add this InstallShield prerequisite to Basic MSI and InstallScript MSI projects.

New FlexNet Connect 11.0.1 Redistributables Available

InstallShield includes support for FlexNet Connect 11.0.1 in Basic MSI and InstallScript MSI projects. Use the Update Notifications view in InstallShield to include one of the two FlexNet Connect 11.0.1 merge modules—one has the Common Software Manager, and the other does not. These merge modules replace the FlexNet Connect 11 merge modules. For details about the changes in the updated merge modules, see the FlexNet Connect release notes.

Additional Changes

For a list of issues that are resolved in InstallShield 2009 SP2, see the release notes. The release notes are available from the Help menu in InstallShield.

What's New in InstallShield 2009 SP1

InstallShield 2009 supports a beta version of Windows Installer 4.5. InstallShield 2009 Service Pack 1 (SP1) includes changes that offer support for the final released version of Windows Installer 4.5:

InstallShield Prerequisites for Windows Installer 4.5

InstallShield now includes the following InstallShield prerequisites for the final released version of the Windows Installer 4.5 redistributables:

- Windows Installer 4.5 for Windows Vista and Server 2008 (x86)
- Windows Installer 4.5 for Windows Vista and Server 2008 (x64)
- Windows Installer 4.5 for Windows Vista and Server 2008 (IA64)
- Windows Installer 4.5 for Windows Server 2003 SP1 and later (x86)
- Windows Installer 4.5 for Windows Server 2003 and XP (x64)
- Windows Installer 4.5 for Windows Server 2003 (IA64)
- Windows Installer 4.5 for Windows XP SP2 and later (x86)

You can add these InstallShield prerequisites to Basic MSI and InstallScript MSI projects if you want Windows Installer 4.5 to be installed at run time.

For more information, see Adding Windows Installer Redistributables to Projects.

Additional Changes

For a list of issues that are resolved in InstallShield 2009 SP1, see the release notes. The release notes are available from the Help menu in InstallShield.

What's New in InstallShield 2009

New Features

InstallShield includes the following new features.

Ability to Associate InstallShield Prerequisites with Features for Chaining Installations

InstallShield now enables you to associate InstallShield prerequisites with one or more features. This new type of InstallShield prerequisite is called a *feature prerequisite*. It is installed if a feature that contains the prerequisite is installed and if the prerequisite is not already installed on the system.

Including InstallShield prerequisites in your project enables you to chain multiple installations together, bypassing the Windows Installer limitation that permits only one Execute sequence to be run at a time. The **Setup.exe** setup launcher serves as a bootstrap application that manages the chaining.

The Redistributables view is where you add InstallShield prerequisites to a project and specify whether you want them to run before your main installation or be associated with one or more features in your main installation.

Previously, all InstallShield prerequisite installations were run before the main installation ran, and the InstallShield prerequisites could not be associated with any features. This type of prerequisite, which is still available, is called a *setup prerequisite*.

Basic MSI projects include support for this feature.

To learn more, see:

- Setup Prerequisites vs. Feature Prerequisites
- Associating an InstallShield Prerequisite with a Feature in a Basic MSI Project
- Run-Time Behavior for an Installation that Includes InstallShield Prerequisites
- Working with InstallShield Prerequisites that Are Included in Installation Projects

Beta Windows Installer 4.5 Support for Installation of Multiple Packages Using Transaction Processing

InstallShield lets you add Windows Installer packages to Basic MSI and InstallScript MSI projects as chained .msi packages. If your Basic MSI or InstallScript MSI installation includes chained .msi packages and Windows Installer 4.5 is present on the target system, the Windows Installer installs the multiple packages using transaction processing. The packages are chained together and processed as a single transaction. If one or more of the packages in the transaction cannot be installed successfully or if the end user cancels the installation, the Windows Installer initiates rollback for all of the packages to restore the system to its earlier state.

The Chained .msi Packages area of the Releases view is where you add to your project one or more .msi packages that you want to be chained to your main installation. This area is also where you can assign release flags to the chained .msi packages, configure settings such as the properties that should be passed to the chained packages, and specify conditions.

For more information, see:

- Configuring Multiple Packages for Installation Using Transaction Processing
- Overview of Multiple-Package Installations that Use Transaction Processing
- Adding a New Chained .msi Package to Your Project
- Chained .msi Package Settings

Beta Windows Installer 4.5 Support for Using the InstallScript Engine as an Embedded User Interface for InstallScript MSI Installations

For InstallScript MSI projects, you now have the option to use the InstallScript engine as an embedded custom user interface (UI) handler, rather than as an external custom UI handler, as it has traditionally been used. Windows Installer 4.5 includes support for this new capability. Use this new embedded option if you want end users to be able to launch your installation directly from an .msi package, but you also want your installation to include a highly customized user interface that you have defined through InstallScript.

To specify whether you want to use the new embedded InstallScript UI functionality or the traditional external InstallScript UI functionality, use the new InstallScript User Interface Type setting; this is a project-wide setting in the General Information view. By default, InstallShield uses the traditional style for all InstallScript MSI projects; a **Setup.exe** setup launcher is required for this traditional option.

A new INSTALLSCRIPTMSIEEUI variable is available. This variable is set to True at run time if the new style is used; otherwise, it is set to False.

Note that the new style does have some limitations that the traditional style does not. For example, some of the InstallScript functions and command-line parameters are not supported or behave differently with the new style. For detailed information about the two styles, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

For additional information, see:

- Specifying the InstallScript User Interface Type for InstallScript MSI Installations
- InstallScript MSI Installation Projects
- INSTALLSCRIPTMSIEEUI

Beta Windows Installer 4.5 Support for Shared Component Patching

InstallShield includes support for the new shared component patching feature that is available with Windows Installer 4.5. The new Multiple Package Shared Component setting in the Components view lets you specify whether you want to enable shared component patching for the selected component. Selecting Yes sets a new component attribute that is designed to prevent Windows Installer from downgrading files when patches that contain components shared across multiple packages are uninstalled. The intent is to keep the highest version of a component present on the target system, even if the patch that contains that highest version is being uninstalled. The default value for this option is No.

Windows Installer 4.5 supports this new functionality; earlier versions of Windows Installer ignore this new setting. In addition, if the DisableSharedComponent policy is set to 1 on a target system, Windows Installer ignores this setting for all packages.

This feature applies to Basic MSI, InstallScript MSI, Merge Module, and Transform projects.

To learn more, see Specifying Whether Shared Component Patching Should Be Enabled for a Component.

Beta Windows Installer 4.5 Support for Superseding Components

Use the new Uninstall Superseded Component setting in the Components view to specify how you want Windows Installer 4.5 to handle the selected component during installation of a superseding patch under certain conditions. To specify that this component in the current patch should be flagged for uninstallation in order to avoid leaving this component orphaned on the target system after a superseding patch is applied, select Yes. If a subsequent patch is installed and it is flagged to supersede the first patch, Windows Installer can unregister and uninstall this component if appropriate. If you select No, the superseding patch can leave an orphaned component on the target system, and none of the feature's remaining components can be maintained. The default value for this setting is No.

Windows Installer 4.5 supports this new functionality; earlier versions of Windows Installer ignore this new setting.

This feature applies to Basic MSI, InstallScript MSI, Merge Module, and Transform projects.

For more information, see the description of the Uninstall Superseded Component setting.

To learn more about patch sequences, see Patch Sequencing.

Beta Windows Installer 4.5 Support for Running a Custom Action Only During Patch Uninstallation

Use the new Run During Patch Uninstall setting for a custom action to indicate whether Windows Installer should run the selected custom action only when a patch is being uninstalled. This setting is available in the Custom Actions and Sequences view of Basic MSI, InstallScript MSI, MSI Database, and Transform projects. It is also available in the Custom Actions view of Merge Module and MSM Database projects.

You can also use the Run During Patch Uninstall check box on the Additional Options panel in the Custom Action Wizard to configure the behavior of a custom action.

For details about this new functionality, see Run During Patch Uninstall.

Beta Support for Windows Installer 4.5 Redistributables

InstallShield includes several InstallShield prerequisite files (.prq) for Windows Installer 4.5.

This feature applies to Basic MSI and InstallScript MSI projects.

For more information, see Adding Windows Installer Redistributables to Projects.

Ability to Create Unicode Versions of the Setup.exe and Update.exe Bootstrappers

InstallShield now enables you to specify whether you want to create a Unicode version or an ANSI version of the **Setup.exe** setup launcher for a Basic MSI project. Previously, if your Basic MSI project included a setup launcher, InstallShield always built an ANSI version; it did not include support for building a Unicode version.

A Unicode setup launcher can correctly display double-byte characters in the user interface of the setup launcher, regardless of whether the target system is running the appropriate code page for the double-byte-character language. An ANSI setup launcher displays double-byte characters in the setup launcher dialogs if the target system is running the appropriate code page. However, it displays garbled characters instead of double-byte characters in those dialogs if the target system is not running the appropriate code page.

Use the new Setup Launcher Type setting on the Setup.exe tab for a release in the Releases view to specify whether you want to use Unicode or ANSI. Unicode is the default type for all new Basic MSI projects.

InstallShield also now enables you to specify whether you want to create a Unicode version or an ANSI version of the **Update.exe** update launcher for a patch or a QuickPatch package.

Use the new Update Launcher Type setting on the Advanced tab for a patch configuration in the Patch Design view to specify whether you want to use Unicode or ANSI. For a QuickPatch project, the Update Launcher Type setting is on the Advanced tab in the Build Settings area of the General Information view. Unicode is the default type for all new patch configurations and QuickPatch packages.

Ability to Create a Log File for the Setup.exe Bootstrapper

A new `/debuglog` command-line parameter is available for the **Setup.exe** setup launcher for Basic MSI and InstallScript MSI projects. Use this command-line parameter to generate a log file for debugging. For more information, see /debuglog.

Support for Managed-Code Custom Actions

InstallShield lets you add managed-code custom actions to Basic MSI, InstallScript MSI, and Merge Module projects. This type of custom action calls a public method in a .NET assembly that is written in managed code such as Visual Basic .NET or C#.

For detailed information, see:

- Calling a Public Method in a Managed Assembly
- Run-Time Requirements for Managed-Code Custom Actions
- Specifying the Signature for a Managed Method in an Assembly Custom Action
- Configuring Custom Action Settings

Support for Installing Multiple Instances of a Product

InstallShield now includes support for creating an installation that allows an .msi package to be used to install multiple instances of a product in the same context on the same machine. Use the new Multiple Instances tab for a product configuration in the Releases view to define different instances of your product and configure the properties that are associated with each instance.

At build time, InstallShield creates a product code–changing instance transform for each instance and streams the instance transforms into the .msi package. At run time, the setup launcher displays a new instance selection dialog that lets end users specify whether they want to install a new instance, or update or maintain an already installed instance.

In addition, if you build a patch in the Patch Design view for a product whose installation includes multiple-instance support, InstallShield now creates a patch that enables end users to update a specific instance or all instances. At run time, the **Update.exe** file displays a patch version of the instance selection dialog.

A new `/instance` command-line option is available for **Setup.exe** and **Update.exe**. This option lets you specify which instance you want to install, update, or uninstall; it also lets you suppress the instance selection dialog.

Multiple-instance support is available for Basic MSI projects.

For more information, see:

- Installing Multiple Instances of Products
- Run-Time Requirements for Multiple-Instance Support
- Configuring Multiple Instances in InstallShield
- Special Considerations for Multiple-Instance Support
- Configuring and Building a Release that Includes Multiple-Instance Support
- Creating Patches for Multiple Instances of a Product
- Run-Time Behavior for Installing Multiple Instances of a Product
- Creating a Setup Launcher
- Multiple Instances Tab for a Product Configuration
- Setup.exe and Update.exe Command-Line Parameters

New Microsoft .NET Redistributables Available

InstallShield now includes many new .NET-related InstallShield prerequisites that you can add to Basic MSI and InstallScript MSI projects:

- Microsoft .NET Framework 3.5 (Web Download)
- Microsoft .NET Framework 3.5 (Full Package)

- Microsoft .NET Framework 3.5 Language Packs (x86, x64, IA64)
- Microsoft .NET Framework 3.0 SP1 (Web Download)
- Microsoft .NET Framework 3.0 Language Packs
- Microsoft .NET Framework 2.0 SP1 (x86, x64, IA64)

In addition, an updated Microsoft .NET Framework object is available in InstallShield for InstallScript projects. This object includes support for versions 3.5, 3.0 SP1, 3.0, 2.0 SP1, 2.0, 1.1 SP1, and 1.0 SP3 of the .NET Framework, including 32-bit, 64-bit x64, and 64-bit Itanium versions. The object also includes all supported language packs, as well as the Web downloader redistributables where available.

For more information, see:

- Adding .NET Framework Redistributables to Projects
- Including the Microsoft .NET Framework and Microsoft .NET Framework Language Pack Prerequisites
- Microsoft .NET Framework Object Wizard

New .msi Package Tools

InstallShield includes several new tools:

- InstallShield MSI Diff enables you to quickly compare two .msi, .msm, or .pcp files. It lets you apply one or more .msp and .mst files to an .msi file and see the changes in the resulting .msi database. You can also use this tool to compare two InstallShield project files (.ism or .ise) that are saved in binary format. This tool uses color coding to show additions, modifications, deletions, and schema differences. You can easily integrate it with most source code control systems.
- InstallShield MSI Query lets you test SQL statements using the Windows Installer version of SQL before you run them in your build script. You can quickly see if a SQL statement is formatted correctly and view the results that it generates.
- InstallShield MSI Sleuth is a diagnostic tool that lets you view the current installed state of a target system. InstallShield MSI Sleuth displays a list of all of the .msi packages that are installed. You can click any .msi package in the list to see the status of its features and components, its known source locations, as well as tables and binary streams within the database. This tool also helps you identify the installed product or products whose packages contain a specific component code.
- InstallShield MSI Grep searches a collection of .msi and .msm packages for specific text. You can narrow the search to show results for only certain tables or columns.

You can launch any of these tools from the InstallShield Tools subfolder on the Windows Start menu.

These InstallShield MSI tools are included with the Premier and Professional editions of InstallShield. The Premier edition also includes a separate installation and extra licenses that let you install just the tools, without InstallShield, on separate machines. For specific terms, see the InstallShield End-User License Agreement.

For more information, see InstallShield MSI Tools.

Ability to Compress Files that Are Streamed into Setup.exe and ISSetup.dll and to Specify the Compression Level

If you build a release that uses a **Setup.exe** setup launcher or a **ISSetup.dll** file (which contains the InstallScript engine), InstallShield now compresses files that it streams into the **Setup.exe** file or the **ISSetup.dll** file. The default compression level that InstallShield uses offers a balance between file size and time that is required to extract the compressed files at run time. If you want to change the compression level or you do not want to use any compression, you can override the default level through a machine-wide setting.

By default, InstallShield does not compress any files that have a .cab file extension when it is streaming them into the **Setup.exe** file at build time, since .cab files are already a compressed type of file. You can modify this default compression exclusion list to include other file types or specific files as needed. The exclusion list is a machine-wide setting.

This feature applies to Basic MSI and InstallScript MSI projects.

For more information, see Configuring the Compression Level for Files that Are Streamed into Setup.exe and ISSetup.dll.

Support for Multi-Part .cab Files

The .cab file type has some limitations. For example, the maximum size of a single .cab file is 2 GB. In addition, some users have had trouble signing large .cab files and verifying the digital signature of large signed .cab files. To work around these limitations, InstallShield now has a new default limit of 600 MB for a .cab file. When InstallShield is creating the .cab files for your release and it reaches the limit, it splits the data into two or more .cab files, creating multi-part .cab files.

You can modify the maximum size limit if necessary. In addition, if you do not want InstallShield to create multi-part .cab files, you can configure it to create single .cab files.

This functionality applies to Basic MSI and InstallScript MSI projects. In addition, it is applicable only if you are building a compressed network image release in which all of the files are embedded in the single-file .msi package or the **Setup.exe** setup launcher. This functionality does not apply to custom compression, where only the files that are associated with one or more features are compressed into .cab files.

For more information, see Configuring the Maximum Size for .cab Files.

Support for Adding an Open Dialog to a Basic MSI Installation to Let End Users Browse to a File

InstallShield includes support for launching the Open dialog from one of the dialogs in your Basic MSI installation. End users click a browse button in one of your dialogs, and this launches the Open dialog. The Open dialog lets the end user browse for a file. When the end user selects the file and clicks the Open button, the Open dialog closes, and the installation writes the full path and file name in an edit field on the dialog. The installation also sets the value of the **IS_BROWSE_FILEBROWSED** property to the path and file name of the file that the end user selected.

To use this functionality, you must add to your project the new Windows Installer DLL custom action called FileBrowse. This custom action calls the **FileBrowse.dll** file. In addition, you must add an edit field control and the browse button that launches the Open dialog, and set a related dialog event. For complete instructions, see Launching a File Open Dialog.

Support for Installing IIS 7 Web Sites on Windows Server 2008

InstallShield now lets you create and manage IIS 7 Web sites, virtual directories, Web service extensions, and application pools on Windows Server 2008 systems. This functionality is available in Basic MSI, InstallScript, and InstallScript MSI projects.

Microsoft SQL Server 2005 Express SP2 Prerequisite Available

InstallShield now includes an InstallShield prerequisite for Microsoft SQL Server 2005 Express Edition SP2. You can add this InstallShield prerequisite to Basic MSI and InstallScript MSI projects.

MySQL 5.0 Support

InstallShield now lists the 5.0.x versions of MySQL in the predefined list of database servers that you can select when you are specifying in the SQL Scripts view the target database servers that your product supports. Previously, it was necessary to create a custom version requirement.

Microsoft Visual Studio 2008 Support

InstallShield is now integrated with Visual Studio 2008, enabling development of installations and products within the same Visual Studio interface.

Ability to Convert Visual Studio Setup and Merge Module Projects to InstallShield Projects

InstallShield now lets you convert a Visual Studio 2008, Visual Studio 2005, Visual Studio .NET 2003, or Visual Studio .NET setup project (.vdproj) to a Basic MSI project (.ism). In addition, InstallShield enables you to convert a Visual Studio 2008, Visual Studio 2005, Visual Studio .NET 2003, or Visual Studio .NET merge module project (.vdproj) to an InstallShield Merge Module project (.ism).

Converting these Visual Studio projects to InstallShield projects enables you to modify the layout of dialogs through a visual Dialog Editor, manage features and components, and use other functionality that is available in InstallShield.

To learn more, see Converting or Importing Visual Studio Projects into InstallShield Projects.

Arabic (Saudi Arabia) and Hebrew Language Support, and Dialog Editor Support for Right-to-Left Languages

InstallShield now includes support for Arabic (Saudi Arabia) and Hebrew languages, which are written and read from right to left. All of the default end-user dialog strings are available in these languages.

Since these languages are read from right to left, InstallShield also includes support for mirroring Arabic and Hebrew dialogs; that is, InstallShield uses a right-to-left layout for Arabic and Hebrew dialogs. Thus, for example, buttons that are on the right side of dialogs in English and other left-to-right languages are moved to the left side of right-to-left-language dialogs. In addition, InstallShield uses mirror-image versions of the dialog images that are displayed for the built-in dialog themes.

The right-to-left layouts and reversed images are used in the Dialog Editor pane in the Dialogs view of InstallShield, and also at run time.

The Arabic and Hebrew support is available in the InstallShield Premier Edition. The Premier edition now includes support for 35 different languages.

The following project types include support for this feature: Basic MSI and Merge Module.

For more information, see Dialog Support for Right-to-Left Languages.

String Entry Validation in InstallScript Files (.rul)

When you build a project that includes an InstallScript file (.rul) and the InstallScript code contains one or more references to string entries that use the string constant operator (@), InstallShield now validates the string entries at build time.

If a string identifier in the project's InstallScript file is not defined for one of the project's string entries, InstallShield displays build warning -7174.

This applies to the following project types: Basic MSI with InstallScript custom actions, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module with InstallScript custom actions.

To learn more, see:

- Description of build warning -7174
- String Constant Operator (@)

New FlexNet Connect 11 Redistributables Available

InstallShield includes support for FlexNet Connect 11 in Basic MSI and InstallScript MSI projects. Use the Update Notifications view in InstallShield to include one of the two FlexNet Connect 11 merge modules—one has the Common Software Manager, and the other does not.

Enhancements

InstallShield includes the following enhancements.

Best Practice Dynamic File Linking

When you add or modify a dynamic file link in your project, you can now specify which component creation method you want InstallShield to use: a new best practice method, or the previously available one-component-per-directory method.

When best practices for component creation are followed, InstallShield creates a separate component for each portable executable (PE) file in the dynamically linked folder and sets each PE file as the key file of its component. If you later want to create a patch that updates one of the dynamically linked PE files, it is easier to do so than it would be if you had used the one-component-per-directory method.

Previously, any time that you added dynamic file links to a project, InstallShield automatically created one component for all of the dynamically linked files at build time. However, if your dynamic file link included PE files, Windows Installer best practices for component creation were not followed.

By default, InstallShield considers the following file types to be PE files: .exe, .dll, .ocx, .vxd, .chm, .hlp, .tlb, and .ax. You can modify this list through the new File Extensions tab on the Options dialog box.

The automation interface includes support for this new best practice method. The ISWiDynamicFileLinking object includes a new CreateBestPracticeComponents property that lets you specify whether you want to use the best practice method, or the previously available one-component-per-directory method for a dynamic file link. When you create a new dynamic file link using the AddDynamicFileLinking method, the best practice method is used by default.

The best practice dynamic file linking applies to Basic MSI, InstallScript MSI, and Merge Module projects.

To learn more, see:

- Determining the Appropriate Component Creation Method for Dynamically Linked Files
- Dynamic File Link Settings Dialog Box
- File Extensions Tab
- ISWiDynamicFileLinking Object

- AddDynamicFileLinking Method

Ability to Hide Setup Prerequisites from the List of Prerequisites to Be Installed

If a target system needs one or more setup prerequisites to be installed, the setup prerequisite dialog is displayed at run time before the main installation runs. The Behavior tab in the InstallShield Prerequisite Editor has a new check box that lets you specify whether you want a setup prerequisite to be hidden from the list of prerequisites in the prerequisite dialog. If a prerequisite is hidden, it is installed when the conditions require it, even though it is not listed as one of the prerequisites that needs to be installed.

New prerequisites and existing prerequisites that were created before this functionality was available are not hidden by default. You can change this behavior by selecting the new check box on the Behavior tab.

For more information, see Specifying Whether to Include the Name of a Prerequisite in the List of Setup Prerequisites to Be Installed on the Target System.

Ability to Show the Progress of an InstallShield Prerequisite Installation at Run Time

The Behavior tab in the InstallShield Prerequisite Editor has a new check box that lets you specify whether you want the prerequisite installation to show installation progress messages from Windows Installer at run time. This augments the progress bar to reflect the current progress status of the .msi installation. This functionality is available only if the prerequisite launches an .msi file; it is not possible if the prerequisite launches a **Setup.exe** file. For more information, see Specifying Whether to Show the Progress of an InstallShield Prerequisite Installation at Run Time.

For new prerequisites and existing prerequisites that were created before this functionality was available, the progress is not shown by default. You can change this behavior by selecting the new check box on the Behavior tab.

Note that if you specify that you want to show the progress, only some of the available command-line parameters are supported. To learn more, see Specifying Command-Line Parameters for an InstallShield Prerequisite.

Ability to Use Windows Installer Properties in Command-Line Statements for InstallShield Prerequisites

Basic MSI and InstallScript MSI installations now can substitute and pass the following properties on setup prerequisite command lines: **ProductLanguage**, **ISPREREQDIR**, **SETUPEXEDIR**, and **SETUPEXENAME**. You can use these properties to identify the launching installation (**"[SETUPEXEDIR]\[SETUPEXENAME]"**), to identify full paths to files that are included in the prerequisite (**"[ISPREREQDIR]myconfig.ini"**), or to pass the selected language to multilingual prerequisites such as an InstallShield setup launcher (**/l[ProductLanguage]**). The new feature prerequisites include command-line support for any Windows Installer property, including the aforementioned properties.

You can specify the command lines on the Application to Run tab of the InstallShield Prerequisite Editor.

For more information, see Specifying Command-Line Parameters for an InstallShield Prerequisite.

New Option for Reboot Behavior of InstallShield Prerequisites

The Behavior tab of the InstallShield Prerequisite Editor is where you specify how an InstallShield prerequisite installation should proceed if it appears that the target machine needs to be restarted. The **If the prerequisite appears to need a reboot** list is where you specify the behavior. This list has a new option, called **Note it, fail to resume if the machine is rebooted, and reboot after the installation**. If it appears that a restart is required at run time but you want to postpone it until the end of the main installation (or until a subsequent prerequisite triggers a restart), select this new option.

For more information, see Behavior Tab.

Support for Single-File, Compressed InstallScript Installations of up to 4 GB

A single-file, compressed InstallScript installation can now be up to 4 GB in size. Previously, when end users ran large single-file, compressed InstallScript installations, the installations crashed during setup initialization.

Note that **Setup.exe** files cannot exceed 4 GB because Windows will not load an executable file that is larger than 4 GB.

Ability to Install IIS Web Sites Without Virtual Directories and to Associate Web Sites with Components

InstallShield now includes support for installing IIS Web sites without any virtual directories. In addition, the General tab that InstallShield displays when you select a Web site in the Internet Information Services view now has a Component setting; use this setting to associate the selected Web site with a component. As a result of these enhancements, a Web site is created on a target machine if a virtual directory or a component that is associated with it is installed.

If a Web site is associated with a component, the Web site's **Delete Web Site on Uninstall** check box corresponds with the Permanent setting for that component in a Basic MSI or InstallScript MSI project, or with the Uninstall setting for that component in an InstallScript project. That is, if you select or clear the **Delete Web Site on Uninstall** check box for a Web site, InstallShield automatically updates the value of the component's Permanent setting or Uninstall setting, as appropriate.

Previously, InstallShield did not include support for associating Web sites with components. Therefore, if a Web site in an installation did not have any virtual directories, the Web site would not be created at run time.

If you add a new Web site through the Internet Information Services view in InstallShield 2009, InstallShield automatically associates that Web site with a component. If you upgrade an InstallShield 2008 or earlier project that already has an IIS Web site, InstallShield does not automatically associate that Web site with a component.

The following project types support these IIS enhancements: Basic MSI, InstallScript, and InstallScript MSI.

Note that if a Web site is associated with a component in an InstallScript project, any virtual directories that are added to that Web site must be associated with the same component. Therefore, if you try to change the component for a Web site that contains one or more virtual directories, InstallShield displays a message box to inform you that it will also make the same component change for all of that Web site's virtual directories; InstallShield also displays this message box if you try to change the component for any virtual directories in a Web site. In either case, the message box enables you to proceed or cancel the component change. For Basic MSI and InstallScript MSI projects, keeping a Web site in the same component as all of the Web site's virtual directories is not required, but it is recommended.

To learn more, see:

- Creating a Web Site and Adding an Application or a Virtual Directory
- Feature and Component Associations for IIS Support
- Uninstalling Web Sites, Applications, and Virtual Directories

Simplification of QuickPatch Packages

InstallShield now offers the ability to build streamlined QuickPatch packages, which typically have fewer new subfeatures and built-in InstallShield custom actions than those built in earlier releases of InstallShield. A new Streamline QuickPatch setting on the Advanced tab in QuickPatch projects lets you specify whether InstallShield should create this new simpler type of QuickPatch package.

For more information, see:

- Specifying Whether to Streamline the QuickPatch Package
- Advanced Tab

Ability to Build a Patch from the Command Line and from MSBuild

The `-patch_config` command-line parameter is available for command-line builds (including the Standalone Build command-line builds) with **ISCmdBld.exe**. Use this parameter to build a patch from the command line.

If you use an .ini file to pass parameters to the command line, you can use the new `PatchConfigName` parameter in the [Project] section of your .ini file to indicate the patch configuration that you want to build.

In addition, the InstallShield task for MSBuild now includes a PatchConfiguration parameter, which you can use to specify the patch configuration that you want to build through MSBuild.

To learn more, see:

- ISCmdBld.exe
- Passing Command-Line Build Parameters in an .ini File
- Microsoft Build Engine (MSBuild)

Ability to Password-Protect Patches and QuickPatch Packages

InstallShield now has new password settings that let you password-protect patches and QuickPatch packages. These settings are on the Advanced tab in the Patch Design view of Basic MSI and InstallScript MSI projects, and on the Advanced tab of QuickPatch projects.

If you password-protect a patch or QuickPatch package, any end user who wants to install the package must enter a case-sensitive password to launch your update.

To learn more, see the following:

- Password-Protecting a Patch Package
- Password-Protecting a QuickPatch Package
- Advanced Tab (for a patch configuration in the Patch Design view)
- Advanced Tab (for a QuickPatch project)

Patch and Upgrade Validation Support in QuickPatch Projects

When you build QuickPatch projects, InstallShield now runs patch and upgrade validation. This validation helps identify some common problems that may be encountered when attempting to upgrade a product with a QuickPatch package.

Previously, the patch and upgrade validation was available only for Basic MSI projects and InstallScript MSI projects, and patches that were created in the Patch Design view.

To learn more, see Validating Upgrades, Patches, and QuickPatch Packages.

Ability to Select Multiple .cub Files for Performing Validation at Build Time

The Validation tab on the Options dialog box in InstallShield now lets you specify more than one .cub file that should be used to validate your installation package or merge module when it is built from within InstallShield.

In addition, you can now pass more than one .cub file through the `-m` command-line parameter for command-line builds (including the Standalone Build command-line builds) with **ISCmdBld.exe**. If you use an .ini file to pass parameters to the command line, you can now specify more than one .cub file for the CubFile parameter in your .ini file.

The RunMsiValidator parameter of the InstallShield task for MSBuild has been enhanced to accept more than one .cub file.

For more information, see:

- Requirements for the Windows Logo Program
- Validating an Installation Package or Merge Module
- Specifying Whether Validation Should Be Performed at Build Time
- Validating an Installation Package or Merge Module
- Validation Tab (on the Options dialog box)
- Build Menu
- ISCmdBld.exe
- Passing Command-Line Build Parameters in an .ini File
- Microsoft Build Engine (MSBuild)

New Validator for the InstallShield Best Practice Suite

ISBP20 is a new validator that is available with the InstallShield Best Practice Suite. ISBP20 verifies that no registry entries contained in the **Registry** table attempt to remove root-level registry keys or other keys that would cause adverse issues on target machines during uninstallation.

The InstallShield Best Practice Suite is available in the Premier edition of InstallShield for Basic MSI, InstallScript MSI, and MSI Database projects.

Ability to Use the /v Command-Line Parameter More than Once to Pass More than One Parameter from Setup.exe to the .msi File

If you want to pass more than one argument from **Setup.exe** to **Msiexec.exe**, you can use the `/v` option multiple times at the command line, once for each argument. Previously, the `/v` option could be used only once, and all parameters were passed through this instance.

The following project types include support for this enhancement: Basic MSI and InstallScript MSI.

For more information, see Setup.exe and Update.exe Command-Line Parameters.

Improved Compatibility Between the Standalone Build and InstallShield

The Standalone Build that is available with InstallShield Premier Edition now uses the same directory structure that InstallShield uses for its program files. Therefore, if you need to copy a redistributable or some other file from a machine that has InstallShield to a machine that has the Standalone Build, use the same relative path. Previously, the directory structures were different and inconsistent.

In addition, the **ISCmdBld.exe** file that is used for command-line builds with InstallShield is now installed with the Standalone Build. Previously, the Standalone Build used **IsSaBld.exe**, a different file, for command-line builds. **ISCmdBld.exe** now supports parameters that previously only **IsSaBld.exe** supported:

- -o <merge module search path>—Specifies one or more comma-delimited folders that contain the merge module (.msm) files referenced by your project.
- -t <Microsoft .NET Framework path>—Specifies the path to the Microsoft .NET Framework. The path is the location of the .NET Framework that is installed on the build machine.
- `-h`—Indicates that you want to skip the upgrade validators at the end of the build.
- `-g <minimum target MSI version>`—Specifies the minimum version of Windows Installer that the installation requires on the target machine.
- `-j <minimum target Microsoft .NET Framework version>`—Specifies the minimum version of the .NET Framework that the installation requires on the target machine.

Also as part of this enhancement, the Standalone Automation Interface uses the same **ISWiAutomation15.dll** file that InstallShield uses, but it is installed to a different location. If you have existing automation scripts that work with the InstallShield Automation Interface, you no longer need to change the library name from IswiAuto*N* to SAAuto*N* throughout the scripts in order to use them with the Standalone Automation Interface. Note that with this change, the ISWiProject object includes support for several properties that were previously available for only the Standalone Automation Interface:

- DotNetFrameworkPath
- MergeModuleSearchPath
- MinimumTargetDotNetVersion
- MinimumTargetMSIVersion
- SelfRegistrationMethod
- SkipUpgradeValidators

An advantage of these compatibility improvements is that only one set of binaries, rather than two separate sets, is built for the Standalone Build and InstallShield; if an InstallShield hotfix or service pack is released, it can be used to update the Standalone Build and InstallShield. Previously, separate hotfixes and service packs were required.

To learn more, see:

- Installing the Standalone Build on a Build Machine
- Adding Redistributables to the Build Machine for the Standalone Build
- Standalone Command-Line Build
- Standalone Automation Interface
- Installing the Standalone Build and InstallShield on the Same Machine
- ISCmdBld.exe
- ISWiProject Object

Support for Browsing to Local, Remote, and Alias SQL Servers in the SQLBrowse Run-Time Dialog

The filter functionality of the SQLBrowse dialogs has been enhanced.

To show only remote servers in the SQL Server browse combo box and list box controls in Basic MSI and InstallScript MSI installations, you can set the new Windows Installer property **IS_SQLSERVER_REMOTE_ONLY**. To show only server aliases in the SQL Server browse combo box and list box controls in Basic MSI and InstallScript MSI

installations, you can set the new Windows Installer property **IS_SQLSERVER_ALIAS_ONLY**. Previously, the only filtering that was available was showing only local servers; this is available if you set the **IS_SQLSERVER_LOCAL_ONLY** property. You can now set any combinations of these properties to display multiple types of servers in the SQL Server browse combo box and list box controls.

To learn more, see Overriding the Default SQL Run-Time Behavior.

Two new InstallScript functions are available for InstallScript projects:

- SQLRTSetBrowseOption—This function lets you specify whether the SQL Server browse combo box and list box controls show local servers, remote servers, server aliases, or a combination of these types.
- SQLRTGetBrowseOption—This function returns the current value of the browse option for the SQL Server browse combo box and list box controls, which can display local servers, remote servers, server aliases, or a combination of these types.

Ability to Continue an Installation If the Minimum SQL Database Server Requirements Are Not Met

The Requirements tab that is displayed when you select a SQL connection in the SQL Scripts view has a new **Allow installation to continue when minimum requirements are not met** check box.

If you select this check box and the minimum database server requirements are not met on a target system, the run time skips the SQL connection and all of its SQL scripts, and continues with the installation.

If you clear this check box and the minimum requirements are not met, the installation does not allow the end user to continue with the rest of the installation. This is the default behavior.

This enhancement applies to Basic MSI, InstallScript, and InstallScript MSI projects.

For more information, see Requirements Tab.

SQL Server Alias Support

The SQL run-time support has been enhanced: installations can now list SQL Server alias names in the SQLBrowse dialog. In addition, the SQLLogin dialogs let end users use an alias name to connect to a SQL Server.

This enhancement applies to Basic MSI, InstallScript, and InstallScript MSI projects.

To learn more, see Adding a New SQL Connection.

Predefined System Searches for the .NET Framework

InstallShield has several new predefined system searches:

- Microsoft .NET Framework 3.5
- Microsoft .NET Framework 3.0 SP1
- Microsoft .NET Framework 3.0
- Microsoft .NET Framework 2.0 SP1
- Microsoft .NET Framework 2.0
- Microsoft .NET Framework 1.1
- Microsoft .NET Framework 1.0

If your installation requires any of these, you can use the System Search view or the Installation Requirements page in the Project Assistant to add these system searches to your project. When end users launch your installation, Windows Installer checks the target system to see if the requirements are met; if they are not met, the installation displays the error message that is defined for the system search.

This enhancement applies to Basic MSI and InstallScript MSI projects.

Expanded Support for the Error Type of Custom Action

The Custom Action Wizard now includes support for a type 19 custom action, which displays a specified error message, returns failure, and ends the installation. Previously, the only way to create this type of custom action was to right-click the Custom Actions explorer in the Custom Actions and Sequences view and then click Error, or to use the Direct Editor to manually enter the custom action's table entry.

The following project types now support the error custom action: Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform. Previously, only Basic MSI and InstallScript MSI projects included support.

Ability to Specify Whether to Digitally Sign Source Files

The Signing tab in the Releases view has a new **Sign files in their original location** check box. This check box lets you specify whether you want InstallShield to sign your original source files or just the files that are built into the release. This check box is also available on the Digital Signature Options panel in the Release Wizard. The check box is cleared by default.

The benefit of selecting this check box for a Basic MSI or InstallScript MSI project is that it helps create one patch that updates both compressed and uncompressed versions of a release that contains originally unsigned files.

Previously, the check box was not available, and InstallShield never enabled you to sign the files in their original location.

The automation interface now includes support for this new digital signing functionality. The ISWiRelease object includes a new **SignFilesInPlace** property that lets you specify whether you want InstallShield to sign your original source files or just the files that are built into the release.

To learn more, see:

- Signing Tab for a Release
- Digital Signature Options Panel
- Digital Signing and Security
- ISWiRelease Object

Improved Static COM Extraction

If you use static COM extraction, InstallShield now uses an MD5 algorithm when generating primary keys for the **Registry** table. Therefore, if the COM data does not change, the primary keys do not change between different versions of a package and when the extracted COM data is refreshed.

Previously, InstallShield used random values for the primary keys that it created during static COM extraction. As a result, if the COM data were refreshed or a patch were built, it was possible for new primary keys to be created, even if the COM data had not changed. In the patch scenario, the COM data would be included in the patch if the primary keys changed. If a patch updated a component with unchanged COM data, the COM data could be removed during patch uninstallation, and this could cause issues in the earlier version of the product.

This enhancement applies to Basic MSI and InstallScript MSI projects.

InstallScript Language Enhancements and New Functionality

Some enhancements have been made to the InstallScript language.

Enhancements for .NET Framework 3.5 and .NET Framework 3.0 SP1 Support

A new FOLDER_DOTNET_35 InstallScript variable is available. This variable stores the path of the .NET Framework 3.0 files.

Two new constants are available for use with the function Is:

- REGDB_KEYPATH_DOTNET_35
- REGDB_KEYPATH_DOTNET_30_SP

You can use the REGDB_KEYPATH_DOTNET_30_SP variable when querying whether SP1—or a later service pack—of the .NET Framework 3.0 is installed. To detect whether the RTM version of the .NET Framework 3.0 is installed, use REGDB_KEYPATH_DOTNET_30.

New GetTempFileNameIS Function that Calls the Windows API GetTempFileName to Create a Temporary File

A new InstallScript function called **GetTempFileNameIS** is available. This function calls the Windows API **GetTempFileName** to create a temporary file and perform related actions.

To learn more, see GetTempFileName.

What's New in InstallShield 2008

New Features

InstallShield includes the following new features.

New End-User Dialog Themes for Basic MSI Projects

Dialog themes are predefined sets of images that give your end-user dialogs a unified and distinctive look. With the click of a button, you can now select one of the available themes for your project, and InstallShield applies that theme to all of the interior and exterior dialogs, as well as the **Setup.exe** initialization dialog, in your project. You can easily preview each dialog from within the Dialogs view to see how it looks with the selected theme.

Some of the themes are available in both the Premier and Professional editions of InstallShield, and some are available in only the Premier edition. For more information, see Available Themes and Corresponding Dialog Sizes.

For more information about this feature, see the Dialog Themes section of this documentation.

Digital Signing Improvements

InstallShield now lets you digitally sign any files—including your product's executable files—in your project at build time. In addition, you can now use a personal information exchange file (.pfx) for digital signatures. All of the following project types support this functionality: Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module.

The new Signing tab in the Releases view is where you specify the digital signature information—including the digital certificate files that a certification authority grated to you—that InstallShield should use to sign your files. The Signing tab is also where you specify which files in your installation should be digitally signed. You can also use the Release Wizard to specify all of the digital signature information.

If you specify a .pfx file for signing, InstallShield uses **SignTool.exe** to sign your files. If you specify an .spc file and a .pvk file, InstallShield uses **Signcode.exe** to sign your files. Using a .pfx file is often the preferred method, since it is more likely to work in many different environments (such as locked build machines). If you specify the digital signature password in InstallShield, you will never see a password prompt if you are using a .pfx file. However, if you are using .spc and .pvk files, a password prompt may be displayed.

Previously, InstallShield included support for signing only the .msi, .hdr, and **Setup.exe** files. In addition, InstallShield allowed you to specify .spc and .pvk files for the digital signature, but not .pfx files.

To learn more, see the following:

- Digital Signing and Security
- Digitally Signing a Release and Its Files at Build Time
- Signing Tab for a Release

InstallShield Best Practice Suite Available in Premier Edition

InstallShield includes a set of validators called the InstallShield Best Practice Suite. The InstallShield Best Practice (ISBP) validators in this suite alert you if your installation violates best-practice guidelines.

This feature is available for Basic MSI, InstallScript MSI, and MSI Database projects.

For more information, see:

- InstallShield Best Practice Suite
- Specifying Which ICEs, ISICEs, ISVICEs and ISBPs Should Be Run During Validation

Support for Internet Information Services (IIS) 7.0 and SSL

InstallShield now includes support for IIS 7.

In addition, InstallShield lets you include an SSL certificate for a Web site in your installation. Including an SSL server certificate enables users to authenticate the Web server, check the validity of the Web content, and establish a secure connection.

For more information, see:

- Internet Information Services View
- Specifying the SSL Certificate for a Web Site
- Determining If a Target System Has IIS 6 or Earlier or the IIS 6 Metabase Compatibility Feature
- Version-Specific Information for IIS Support in InstallShield

New Microsoft .NET Prerequisites Available

InstallShield now includes many new .NET-related setup prerequisites that you can add to Basic MSI and InstallScript MSI projects:

- .NET Framework 2.0 (x64)
- .NET Framework 2.0 (x64) Language Packs
- .NET Framework 2.0 (IA64)
- .NET Framework 2.0 (IA64) Language Packs

- .NET Framework 3.0 (x64)

For more information, see Adding .NET Framework Redistributables to Projects.

Updated Microsoft .NET Object for InstallScript Projects

An updated Microsoft .NET object is available in InstallShield. This object includes support for versions 1.0 (SP3), 1.1 (SP1), 2.0, and 3.0 of the .NET Framework, including 32-bit, 64-bit x64, and 64-bit Itanium versions. The object also includes all supported language packs, and the latest service packs of 1.0 and 1.1.

In addition, the object launches and completes the .NET Framework installation as the feature containing the .NET object installs. This allows the .NET Framework to be available as early as possible, in case it is needed to install or configure files that are subsequently installed during the installation.

For more information, see Microsoft .NET Framework Object Wizard.

Visual C++ 8.0 SP1 Merge Modules Available

InstallShield now includes Visual C++ 8.0 SP1 merge modules (version 8.0.50727.762).

Support for the UAC Shield Icon on Dialog Buttons (Basic MSI Projects)

In the Dialog Editor of Basic MSI projects, a new Show UAC Shield Icon property is available for all button controls. If you select True for this property, the User Account Control (UAC) icon is displayed on the button when end users run the installation on Windows Vista systems. If you are using InstallShield on a Windows Vista system, you can see the shield icon on the button in the Dialog Editor as it will be displayed at run time. The shield icon signals to end users that elevated privileges may be required.

For any new Basic MSI projects that you create, the Show UAC Shield Icon property is set to True for the Install button on the ReadyToInstall dialog. If you upgrade a Basic MSI project that was created with InstallShield 12 or earlier to InstallShield 2008, the default value for the Install button's Show UAC Shield Icon property is False. You can override the value for this button, or for any other buttons, as required.

Ability to Require End Users to Scroll Through the EULA in the LicenseAgreement Dialog

InstallShield includes support for disabling the Next button on the LicenseAgreement dialog until the end user reaches the end of the End-User License Agreement (EULA) text in the scrollable EULA control through mouse or keyboard scrolling.

The end user must also select the **I accept the terms in the license agreement** option before the Next button is enabled; this behavior is the same as with earlier releases of InstallShield.

The scroll requirement is not available in the LicenseAgreement dialog by default. To use this functionality, you must add to your project the new Windows Installer DLL custom action called WatchScroll. This custom action calls the **EulaScrollWatcher.dll** file. In addition, you must modify the Next button's Control conditions and add an event to the Memo control.

This is available for Basic MSI projects.

For detailed instructions, see Requiring End Users to Scroll Through the EULA in the LicenseAgreement Dialog.

Microsoft SQL Server 2005 Express SP1 Setup Prerequisite Available

InstallShield now includes a setup prerequisite for Microsoft SQL Server 2005 Express Edition SP1. You can add this setup prerequisite to Basic MSI and InstallScript MSI projects.

Updated DirectX 9.0c Objects

Two DirectX 9.0c objects are available with InstallShield: a Windows Installer–based object that is available for Basic MSI and InstallScript MSI projects, and an InstallScript-based object that is available for InstallScript projects. Both of these objects install all of the latest DirectX 9.0c core and optional components, including 32-bit-specific and 64-bit-specific components.

In addition, some changes have been made to the DirectX 9 Object Wizard for Basic MSI and InstallScript MSI projects. The wizard now lets you specify whether the redistributable files should be included in the Disk1 folder or streamed into the .msi file. This change enables you to use the DirectX 9 object in compressed installations. Also, you can now use the DirectX 9 object in silent installations.

For Basic MSI and InstallScript MSI projects, the custom action that launches the DirectX installation is now sequenced in the Execute sequence and run in deferred system context so that it can be run with elevated privileges on Windows Vista systems.

To learn more, see:

- Including the DirectX 9.0 Object
- DirectX Object Wizard

DIFx 2.1 Support

InstallShield includes support for the latest version of Driver Install Frameworks for Applications (DIFx). This new version, which includes the latest binary files from Microsoft, is available for any Basic MSI, InstallScript, or InstallScript MSI projects that you create in InstallShield. This new version can be installed on Windows Vista systems. Earlier versions of InstallShield included an earlier version of DIFx that failed to install on Windows Vista in some cases.

Support for 64-Bit Self-Registration of COM Servers (Basic MSI Projects)

InstallShield now includes support for 64-bit self-registration of COM servers in Basic MSI projects. If you mark a component as 64 bit and then add a file to that component, you can select the file's Self Register check box to enable 64-bit self-registration of that file during installation. In addition, InstallShield also supports 64-bit self-registration of dynamically linked COM servers. For more information, see:

- Targeting 64-Bit Operating Systems with Basic MSI and InstallScript MSI Installations
- File Properties Dialog Box
- Adding Dynamic File Links to Components

Expanded Operating System Condition Settings in the Setup Prerequisite Editor

The Setup Prerequisite Editor now enables you to specify more details about operating system requirements when you are creating conditions for a prerequisite. The Prerequisite Condition dialog box, which is displayed when you are adding or modifying a condition for a prerequisite through the Setup Prerequisite Editor, lets you select a predefined operating system or select the custom option. With the new custom option, you can configure settings for operating system requirements such as platform, major and minor versions, processor architecture (64-bit or 32-bit), and service pack. If a target system does not meet the operating system requirements, the prerequisite is not installed.

For more information, see the following:

- Adding an Operating System Condition for an InstallShield Prerequisite

- Prerequisite Condition Dialog Box
- Conditions Tab

Ability to Target Windows Server 2008 Systems

InstallShield enables you to specify that your installation requires Windows Server 2008. It also lets you build Windows Server 2008–related conditions for features and components.

Note that Windows Vista and Windows Server 2008 use the same major and minor version numbers. Therefore, if you want to use InstallScript to distinguish between Windows Server 2008 and Windows Vista, check whether SYSINFO.nOSProductType = VER_NT_WORKSTATION; for Windows Vista, this is TRUE; for Windows Server 2008, it is FALSE. For more information, see SYSINFO.

New MSXML 6 SP1 Setup Prerequisites Available

InstallShield now includes some new MSXML setup prerequisites that you can add to Basic MSI and InstallScript MSI projects:

- MSXML 6.0 SP1
- MSXML 6.0 SP1 (IA64)
- MSXML 6.0 SP1 (x64)

To learn more about MSXML, see Run-Time Requirements for XML File Changes.

FlexNet Connect Support

You can add a redistributable for FlexNet Connect 6.1 to Basic MSI and InstallScript MSI projects.

The Update Notifications view in Basic MSI and InstallScript MSI projects lets you select which version of FlexNet Connect you want to include in your project. You can include version 6.1 or any version that is installed in any of the locations that are specified in the Merge Module Location area on the Merge Module tab of the Options dialog box.

The Update Notifications view includes a new Vendor Database setting, which FlexNet Connect 6.1 supports.

Enhancements

Usability Enhancements for Releases

The release settings are now organized by category on several different tabs in the Releases view.

In addition, you can now select Compressed or Uncompressed for the Compression setting from within the Releases view. Previously, the only way to modify this setting was to use the Release Wizard. Note that if you want to specify a custom compression setting (one .cab file per feature or one .cab file per component), you must still use the Release Wizard. The Compression setting is available in Basic MSI, InstallScript MSI, and Merge Module.

The settings in the Distribution view have been moved to the new Postbuild tab in the Releases view for Basic MSI, InstallScript MSI, and Merge Module projects. The Postbuild tab lets you configure settings for distributing releases to a folder or FTP site automatically at build time.

A new Distribute command is available when you right-click a release in the Releases view for Basic MSI, InstallScript MSI, and Merge Module projects. When you select this command, InstallShield copies all of the relevant files for your release to the locations that are specified on the Postbuild tab. Note that for InstallScript and InstallScript Object projects, InstallShield automatically copies the release to the locations that you specify on the Postbuild tab every time that you build the release.

To learn more, see:

- Build Tab for a Release
- Setup.exe Tab for a Release
- Signing Tab for a Release
- .NET/J# Tab for a Release
- Internet Tab for a Release
- Events Tab for a Release
- Distributing Releases to a Folder or FTP Site Automatically

Usability Enhancements for Custom Actions and Sequences

The Custom Actions view and the Sequences view have been combined into a more robust view called the Custom Actions and Sequences view. The combined view supports drag-and-drop editing and copying:

- To sequence a new custom action, drag it from the Custom Actions explorer to the appropriate location in a sequence under the Sequences explorer.
- To move a dialog, standard action, or custom action to a different point in a sequence (or from one sequence to another), drag it from the old location to the new location.
- To copy a custom action from one sequence to another, press and hold CTRL while dragging the custom action from one sequence to another sequence.

The Custom Actions and Sequences view is available in Basic MSI, InstallScript MSI, MSI Database, and Transform projects.

For more information, see:

- Custom Actions and Sequences View (or Custom Actions View)
- Creating Custom Actions in the Custom Actions and Sequences View (or the Custom Actions View)
- Inserting Actions into Sequences
- Copying a Custom Action from One Sequence to Another
- Reordering a Sequence

Usability Enhancements for the Files and Folders view, the Registry View, and the Redistributables View

Several enhancements are available for the Files and Folders view:

- You can right-click a file in the **Destination computer's files** pane and then click the new Open Containing Folder command. Doing so opens a Windows Explorer window and displays the folder that contains the file that you right-clicked.
- A new Add command is available when you right-click the **Destination computer's files** pane. Use this command to display an Open dialog box that lets you browse to the file that you want to add to your project.
- The upper-right corner of this view has a new link (either Show Source Panes or Hide Source Panes). Use this new link to show or hide the two top panes—the **Source computer's folders** pane and the **Source computer's**

files pane—in this view. You can hide the two panes, open a Windows Explorer window, and drag and drop files from the Windows Explorer window to the two remaining panes in InstallShield.

The Registry view also has a new link (either Show Source Panes or Hide Source Panes) in the upper-right corner. Use this new link to show or hide the two top panes—the **Source computer's folders** pane and the **Source computer's files** pane—in this view.

Two enhancements have also been made to the Redistributables view for Basic MSI and InstallScript MSI projects:

- The right pane in this view shows details about the merge module, object, or setup prerequisite that is selected in the upper-left pane. You can now hide or show the details pane by clicking the Show Details or Hide Details link in the upper-right corner of this view.
- The Details pane that is displayed for setup prerequisites now shows complete information about the selected setup prerequisite. This includes conditions, command-line parameters, and other information that is configured for the prerequisite.

Automation Interface Enhancements

Many enhancements have been made to the automation interface.

CreateProject Method Now Enables Creation of Merge Module Projects

The CreateProject method for the ISWiProject object now enables you to create merge module projects. Previously, this method supported only Basic MSI, InstallScript, InstallScript MSI, and InstallScript Object projects.

To learn more, see CreateProject Method.

Support for Configuring Dynamic File Links

The automation interface includes a new object and a new collection for dynamic file links. In addition, the ISWiComponent object includes two new methods and a property that let you add (AddDynamicFileLinking) and remove (RemoveDynamicFileLinking) a component's dynamic file link, as well as get the collection of dynamic file links (ISWiDynamicFileLinkings).

For more details, see:

- ISWiDynamicFileLinking Object
- ISWiDynamicFileLinkings Collection
- ISWiComponent Object
- AddDynamicFileLinking Method
- RemoveDynamicFileLinking Method

Support for Configuring Path Variables

The automation interface includes a new object and a new collection for configuring path variables in a project. In addition, the ISWiProject object includes two new methods and a property that let you add (AddPathVariable) and remove (DeletePathVariable) a path variable from a project, as well as get the collection of path variables (ISWiPathVariables).

To learn more, see:

- ISWiPathVariable Object

- ISWiPathVariables Collection
- ISWiProject Object
- AddPathVariable Method
- DeletePathVariable Method

Support for Modifying String-Table Entries

The automation interface includes new objects and collections for configuring languages and string entries in a project. The ISWiLanguage object includes two methods and a property that let you add (AddStringEntry) and remove (DeleteStringEntry) a string entry from a project, as well as get the collection of string entries (ISWiStringEntries). In addition, the ISWiProject object includes a new property (ISWiLanguages) that gets the collection of languages included in the current project.

For more information, see:

- ISWiLanguage Object
- ISWiLanguages Collection
- ISWiStringEntry Object
- ISWiStringEntries Collection
- AddStringEntry Method
- DeleteStringEntry Method
- ISWiProject Object

Support for Configuring Environment Variables

The automation interface includes a new object and a new collection for environment variables. In addition, the ISWiComponent object includes two new methods and a property that let you add (AddEnvironmentVar) and remove (RemoveEnvironmentVar) a component's environment variables, as well as get the collection of environment variables (ISWiEnvironmentVars).

- ISWiEnvironmentVar Object
- ISWiEnvironmentVars Collection
- ISWiComponent Object
- AddEnvironmentVar Method
- RemoveEnvironmentVar Method

ISWiProject Object Properties for Setting the Company Name, Company URL, and Company Phone Number

The ISWiProject object includes several new properties that let you specify values for General Information view settings:

- CompanyName
- CompanyURL
- CompanyPhone

ISWiRelease Object Properties for Digitally Signing Files

The ISWiRelease object includes several new properties that enable you to configure settings for digitally signing files for releases at build time. The new properties are:

- CertificatePassword
- SignFiles
- SignFilesExclude
- SignFilesInclude
- SignSignedFiles

ISWiRelease Object Properties for Specifying Whether to Keep Unused Directories in the Directory Table

The ISWiRelease object includes a new KeepUnusedDirectories property that lets you specify whether you want to want InstallShield to remove unused directories from the **Directory** table of the .msi file when you build this release.

ISWiRelease Object Property for Configuring Whether to Postpone Any Reboot for Installing or Updating the .NET Framework

The ISWiRelease object includes a new DotNetDelayReboot property that lets you specify whether you want to postpone any reboot associated with installing or updating the .NET Framework on the target system until after your installation has completed.

ISWiRelease Object Property for Specifying Whether to Display a Message Box Asking the End User if They Want to Install the .NET Framework

The ISWiRelease object includes a new DisplayDotNetOptionDialog property. Use this property to specify if a message box should be displayed at run time to let end users indicate whether the .NET Framework should be installed.

ISWiRelease Object Properties for Configuring Build-Time Distribution Options

The ISWiRelease object now includes several new properties that enable you to configure settings for distributing releases to a folder or FTP site automatically at build time. The new properties are:

- DistributeLoc
- DistributeToURLLoc
- DistributeToURLUserName
- DistributeToURLPassword
- DistributeAfterBuild

New RemoveSequenceRecord Method for the ISWiSequence Collection

The ISWiSequence collection has a new RemoveSequenceRecord method that enables you to delete an item from a sequence. To learn more, see RemoveSequenceRecord Method.

Basic MSI Projects Now Support Adding and Removing Support Files

The automation interface now includes support for the ISWiSetupFile and ISWiAdvancedFile objects in Basic MSI projects. These objects include methods (AddSetupFile, DeleteSetupFile, AddAdvancedFile, and DeleteAdvancedFile) that are now available in Basic MSI projects. Previously, these objects and methods were available in only InstallScript, InstallScript MSI, and InstallScript Object projects.

To learn more, see:

- ISWiAdvancedFile Object
- ISWiAdvancedFiles Collection
- AddAdvancedFile Method
- DeleteAdvancedFile Method
- ISWiSetupFile Object
- ISWiSetupFiles Collection
- AddSetupFile Method
- DeleteSetupFile Method
- ISWiProject Object

XML File Change Enhancements

Support for XML file changes has been expanded in InstallShield:

- Now you can test just the XML file changes that are configured for your project through the XML File Changes view without having to build and run your entire installation.
- The XML File Changes view now supports namespaces in XML files.
- InstallShield lets you specify the XML encoding of an XML file.

To learn more, see the following areas:

- Testing Installation Changes to an XML File
- Testing Uninstallation Changes to an XML File
- Using Namespaces in XML Files
- Declaring Namespace Mappings for an XML File
- Adding a Namespace Prefix to an Element
- XML File Changes View
- Namespace Tab for an XML File

To learn more about XML file changes, see the Modifying XML Files section of the InstallShield Help Library.

Faster Direct Editor, String Table Editor, and Files Subview

Loading records in the Direct Editor and the String Table editor in InstallShield takes less time now. In addition, for projects that contain a large number of files, InstallShield now displays the files in the Files subview within the Components view more quickly than in earlier releases.

Enhanced User Account Control Support for InstallScript Projects

The Required Execution Level setting is now available on the **Setup.exe** tab in the Releases view for InstallScript projects. Use this setting to specify the minimum level required by your installation's **Setup.exe** file for running the installation (the setup launcher) on Windows Vista platforms. InstallShield adds a manifest that specifies the required level.

Previously, InstallShield always included a Highest Available manifest for InstallScript projects, and the Required Execution Level setting was available in only Basic MSI and InstallScript MSI projects.

For more information, see Specifying the Required Execution Level for Your Setup Launcher on Windows Vista and Later Platforms.

Shortcuts View Enhancements

Some enhancements have been made to the Shortcuts view for Basic MSI, InstallScript MSI, and merge module projects.

- To change the icon that is used for a shortcut, you can right-click the shortcut and then click the new **Change Shortcut icon** command. InstallShield opens the Change Icon dialog box, which enables you to select the icon file and associated icon index that should be used when the shortcut is created on target systems at run time.
- Shortcuts that are listed in the Shortcuts explorer now show the icon image that will be used on the target system. Previously, the Shortcuts explorer used a different image for all types of shortcuts, even if an icon was specified for the shortcut.

For more details about these enhancements, see:

- Specifying the Icon for a Shortcut
- Shortcuts View

Windows Vista and Internet Explorer 7 Support for One-Click Install Installations

One-Click Install installations in InstallScript projects can now be used on Windows Vista systems and with Internet Explorer 7. The One-Click Install setup player is now an external .ocx file, instead of being embedded in the **Setup.exe** file. The setup player downloads and then launches the **Setup.exe** file with the appropriate command line. This enables end users who are using Windows Vista with limited privileges to run the installation; if elevated privileges are required because of the required execution level specified in the installation's manifest, the appropriate User Account Control (UAC) prompt is displayed when the **Setup.exe** file is launched.

To support this new behavior, a new Generate One-Click Install setting is available on the Internet tab in the Releases view and on the Internet Options panel in the Release Wizard. If you select Yes for this setting, InstallShield includes an .ocx file for the installation.

To learn more about One-Click Install installations, see the Typical Web Installation vs. One-Click Install section of the documentation. In addition, see the following:

- Internet Tab for a Release
- Internet Options Panel

Enhanced Support for the SecureCustomProperties Property

If you set a public property in the user interface sequence of an installation that requests elevated privileges for the execute sequence, and you want to pass the property's value to the execute sequence, the property must be listed as a value for the **SecureCustomProperties** property, or it must be a restricted public property.

InstallShield now automatically adds to the **SecureCustomProperties** property properties that may need to be passed from the user interface sequence to the execute sequence. To learn more, see Specifying that a Public Property Should Be a Restricted Public Property.

This support applies to Basic MSI, InstallScript MSI, and Merge Module projects.

Automatic Downgrade Prevention Entries in Basic MSI and InstallScript MSI Projects

To prevent end users from being able to install the current version of your product over a future major version of the same product, the following conditions should be met: The Upgrades view should contain a major upgrade item, the major upgrade item should be properly configured to prevent the current version of your product from being installed over a future version, and your project should include a properly configured and scheduled type 19 custom action.

When you create a new Basic MSI or InstallScript MSI project, InstallShield automatically adds support for preventing the current installation from overwriting a future major version. To learn more, see the following:

- Preventing the Current Installation from Overwriting a Future Major Version of the Same Product
- ISICE19

Changes for ALLUSERS and for the CustomerInformation Dialog

Beginning with InstallShield 2008, the **ALLUSERS** property is set to 1 by default in all new Basic MSI projects. This is the recommended implementation, since most installations must be run in a per-machine context with administrative privileges.

If you upgrade a project that was created with InstallShield 12 or earlier to InstallShield 2016, InstallShield does not automatically change the value of the **ALLUSERS** property or add this property if it was not defined in the earlier project.

Also new with InstallShield 2008, by default, the CustomerInformation dialog in all new Basic MSI projects does not display the radio button group that enables end users to specify whether they want to install the product for all users or for only the current user. This is the recommended implementation for this dialog.

If you upgrade a project that was created with InstallShield 12 or earlier to InstallShield 2016, InstallShield does not automatically change the CustomerInformation dialog.

To learn more, see:

- Per-User vs. Per-Machine Installations
- ALLUSERS

Ability to Change the Product Version from the Command Line or Through an MSBuild Task Parameter

The -y command-line parameter is available for command-line builds with **ISCmdBld.exe** and **IsSaBld.exe**. Use this parameter to specify a product version from the command line.

In addition, the InstallShield task for MSBuild now includes a ProductVersion parameter, which you can use to specify the product version through MSBuild. This parameter is exposed as the property InstallShieldProductVersion when the default targets file is used.

Using the -y command-line parameter or the InstallShield task ProductVersion parameter is especially helpful if you want to increment the build version (the third field) of the product version.

To learn more, see:

- ISCmdBld.exe
- Standalone Command-Line Build
- Microsoft Build Engine (MSBuild)

Ability to Override Windows Installer Property Values from the Command Line or Through an MSBuild Task Parameter

The -z command-line parameter is available for command-line builds with **ISCmdBld.exe** and **IsSaBld.exe**. Use this parameter to override the value of a Windows Installer property or create the property if it does not exist.

In addition, the InstallShield task for MSBuild now includes a PropertyOverrides parameter, which you can use to override the value of a Windows Installer property or create the property if it does not exist. This property is exposed as the InstallShieldPropertyOverrides ItemGroup passthrough to the PropertyOverrides property on the InstallShield task.

To learn more, see:

- ISCmdBld.exe
- Standalone Command-Line Build
- Microsoft Build Engine (MSBuild)

Windows Installer Properties Used for IIS Data Are Stored in the Registry by Default

Installations that install IIS Web sites and use Windows Installer properties to dynamically set IIS data at run time now write the property and its value to the following registry key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\InstallShield Uninstall Information\{ProductCode}

This change was made to make the value available during uninstallation and repair. Therefore, if your Web site is installed to an end user–specified site number, the Web site and its virtual directories can be successfully uninstalled from that site number. If you do not want the IIS data to be stored in the registry, set the **IS_IIS_DO_NOT_USE_REG** property in your project.

This change applies to Basic MSI and InstallScript MSI projects.

New Setting for Specifying Whether an IIS Web Server Should Allow the CMD Command to Be Used for SSI #exec Directives

You can configure an IIS Web server to prevent the CMD command for the #exec directive from being used to execute shell commands, or you can configure it to allow the CMD command to be used to execute this type of command. The SSIEnableCmdDirective registry value for the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters registry key is what determines whether the CMD command is permitted.

The Internet Information Services view in InstallShield includes a new SSIEnableCmdDirective registry value setting. This setting lets you specify how your installation should configure the SSIEnableCmdDirective registry value on target systems. This setting also lets you specify that the SSIEnableCmdDirective registry value should not be changed at installation run time; this is the default behavior.

For more information, see Specifying Whether a Web Server Should Allow the CMD Command to Be Used for SSI #exec Directives.

New Host Header Name Setting for IIS Web Sites

You can now use the new Host Header Name setting on the Web Site tab for a Web site in the Internet Information Services view to specify the host header name that identifies the IIS Web site that is installed during your installation. Previously, it was necessary to configure the ServerBindings property on the Advanced tab in this view in order to specify the host header name.

For more information, see Specifying the IIS Host Header Name for a Web Site.

Ability to Install an IIS Web Site and Its Virtual Directories to the Next Available New Site Number

InstallShield provides support for installing an IIS Web site and its virtual directories to the next available new site number. The implementation is slightly different, depending on which project type you are using. For more information, see Configuring the TCP Port and Site Numbers.

Ability to Specify Whether New SQL Connections Should Share the Same Windows Installer Properties

A new SQL Scripts tab on the Options dialog box lets you specify how InstallShield should create new database connections by default—either by using the same Windows Installer properties that were used by default for the first connection in your project, or by using a unique set of new Windows Installer properties.

This applies to the following project types: Basic MSI and InstallScript MSI.

For more information, see the following:

- Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties
- SQL Scripts Tab

SQLLogin Dialog Enhancements

The SQLLogin dialog now includes a new control that enables end users to specify the name of the target database catalog. This dialog also has a Browse button next to the new control; the Browse button lets end users select from the list of database catalogs that are available on the target database server.

If you upgrade a Basic MSI project that includes SQL support from InstallShield 12 or earlier to InstallShield 2016, you need to manually import the **SQLLogin.isd** and **SQLBrowse.isd** dialogs into your project to use the new version of the SQLLogin dialog. The .isd files are installed in the following location: *`InstallShield Program Files Folder`***\Support**. To use this updated SQLLogin dialog in InstallScript and InstallScript MSI projects, replace the **SQLServerSelectLogin** function call in your InstallScript code with a call to the new **SQLServerSelectLogin2** function.

Database Import Wizard Enhancements

The Database Import Wizard has a new Advanced Scripting Options panel that lets you specify security script options such as script database, script database users and database roles, script SQL Server logins, and script object-level permissions.

For more information, see Advanced Scripting Options Panel.

New Windows Installer Property for Specifying SQL Connections That Should Not Be Installed or Uninstalled

InstallShield includes support for a new Windows Installer property called **IS_SQLSERVER_CXNS_ABSENT_FROM_INSTALL**. Use this property to specify one or more SQL connections that should be skipped during installation or uninstallation. To specify more than one SQL connection, separate each with a semicolon (;). To skip all of the SQL connections, set the value of this property to **ALL**. Using this property is helpful if you cannot uninstall a product because of a SQL scripting error.

For details about SQL-related properties, see Overriding the Default SQL Run-Time Behavior.

Ability to Remove Unreferenced Directories from the .msi File

The Build tab in the Releases view includes a new Keep Unused Directories setting. Use this setting to specify whether you want InstallShield to remove unused directories from the **Directory** table of the .msi file when you build the selected release. The default value is No.

This setting is available for Basic MSI, InstallScript MSI, and Merge Module projects.

For more information, see Build Tab for a Release.

Ability to Specify COM+ Component File Destinations from Within the Component Services View

The Component Services view has two new Destination fields on the Installation tab: one for a server type of installation, and one for a proxy type of installation. Use these fields to specify the target destination of the selected COM+ component files. Previously, the only way to specify the destination of the components associated with a COM+ application was in the Components view or the Setup Design view.

New Windows Installer Property for Specifying COM+ Applications to Be Installed After the InstallFinalize Action

InstallShield includes support for a new Windows Installer property called **IS_COMPLUS_INSTALL_AT_FINALIZE**. Use this property to specify one or more COM+ applications that should be installed by the ISComponentServiceFinalize action, which is called after the InstallFinalize action. To specify more than one COM+ application, separate each with a semicolon (;). To specify that all COM+ applications should be installed by the ISComponentServiceFinalize action, set the value of this property to **ALL**.

Using this property is helpful if you want to install a COM+ application that contains .NET assemblies to be installed to the GAC because Windows Installer does not commit the changes made in the in-script session to the GAC until the InstallFinalize action.

Additional Predefined System Searches for Basic MSI and InstallScript MSI Projects

InstallShield has several new predefined system searches:

- Adobe Reader 7
- Adobe Reader 6
- Internet Explorer 7.0

If your installation requires any of these products, you can use the System Search view or the Installation Requirements page in the Project Assistant to add these system searches to your project. When end users launch your installation, Windows Installer checks the target system to see if the requirements are met; if they are not met, the installation displays the error message that is defined for the system search.

New and Enhanced Setup.exe and Update.exe Command-Line Parameters (InstallScript Projects)

Two new command-line parameters are available for **Setup.exe** and **Update.exe**:

- `/installfromweb`
- `/media_path`

In addition, the `/L` parameter now supports both hex and decimal language values.

To learn more, see Setup.exe and Update.exe Command-Line Parameters.

Downloader Web Release Type Supports More Location Options for .cab Files

The Downloader type of Web release now lets you specify whether InstallShield should create external .cab files that are downloaded at installation time as needed. Previously, .cab files were always streamed into the .msi package for the Downloader Web release type.

The new external .cab file options are available on the Downloader Options panel of the Release Wizard. This panel is displayed in the Release Wizard if the media type is Web and the Web type is Downloader. The Downloader Options panel has a new Create external .cab files check box. If you clear this check box, the behavior is the same as it was previously: the .cab files are streamed into the .msi package. If you select this check box, you can specify how .cab files should be created: one .cab file per feature, one .cab file per component, or multiple .cab files based on a particular size that you specify.

Downloader and Install from the Web Types of Web Releases Embed All Transforms

The Downloader type of Web release and the Install from the Web type of Web release now always embed all .mst and .ini files in the **Setup.exe** file.

Enhancements for Patch Display Information

The Identification tab, which was previously called the Uninstall tab, is where you specify information that should be displayed for a patch in Add or Remove Programs on systems running Windows Installer 3.0 or later. This tab in the Patch Design view of Basic MSI and InstallScript MSI projects and in the General Information view in QuickPatch projects has settings for items such as the display name, the manufacturer name, and the support URL. Now every time that you change the latest setup for a patch configuration in the Patch Design view or in a QuickPatch project, InstallShield uses the Add or Remove Programs information from the latest setup as the values for the Identification tab settings. You can override the values on the Identification tab as needed.

In addition, the **Allow Patch to Be Uninstalled (Requires Windows Installer 3.0)** check box is now available on the Common tab. This setting was previously available on the Uninstall tab.

For more information, see:

- Common Tab (in the Patch Design view)
- Identification Tab (in the Patch Design view)
- Common Tab (in a QuickPatch project)
- Identification Tab (in a QuickPatch project)

Ability to Specify the Minimum Initialization Time

InstallShield includes a new Minimum Initialization Time setting on the **Setup.exe** tab for a release in the Releases view. Use this setting to specify the minimum number of seconds that the installation should display the initialization dialog—as well as the splash screen, if one is included—when end users run this release.

InstallScript Language Enhancements and New Functionality

Several enhancements have been made to the InstallScript language.

New LaunchApplication and WaitForApplication Functions to Supersede LaunchAppAndWait for Launching Applications with Elevated Privileges

The **LaunchApplication** function uses either the Windows API function **CreateProcess** or the Windows API function **ShellExecuteEx** to launch the specified application. After the application is launched, the installation can optionally call the new **WaitForApplication** function to wait for the application to terminate.

The **WaitForApplication** function waits for a running application, and optionally all child applications that are launched by the running application, to terminate before returning.

Note that calling **LaunchAppAndWait** now calls the following:

```
LaunchApplication( szProgram, szCmdLine, "", LAAW_STARTUPINFO.wShowWindow, LAAW_PARAMETERS.nTimeOut,
nOptions | LAAW_OPTION_CHANGEDIRECTORY | LAAW_OPTION_FIXUP_PROGRAM );
```

The new **LaunchApplicationInit** function is available. This function, which was added to match the naming convention of the **LaunchApplication** function, has the same behavior as the **LaunchAppAndWaitInitStartupInfo** function.

For more information, see:

- LaunchApplication
- WaitForApplication
- LaunchAppAndWait
- LaunchApplicationInit

Several new predefined constants are available:

- LAAW_OPTION_CHANGEDIRECTORY
- LAAW_OPTION_FIXUP_PROGRAM
- LAAW_OPTION_USE_SHELLEXECUTE
- LAAW_OPTION_WAIT_INCL_CHILD

In addition, the LAAW_OPTION_NO_CHANGEDIRECTORY is now obsolete. Passing this parameter has no effect.

The following script variables are also now available:

- LAAW_SHELLEXECUTEINFO
- LAAW_SHELLEXECUTEVERB

New SdRMFilesInUse Dialog and OnRMFilesInUse Event Handler for Minimizing Reboots Through the Restart Manager Infrastructure in InstallScript MSI Projects

A new **SdRMFilesInUse** function is available. This function displays a dialog that includes a list box containing a list of the applications that are open and are locking files. The dialog also includes two radio buttons that allow end users to specify whether the installation should attempt to use the Restart Manager to shut down the applications that are locking files or overwrite the locked files (which most likely results in the need for a reboot to complete the installation).

For InstallScript MSI projects, the new OnRMFilesInUse event handler displays the new **SdRMFilesInUse** dialog. This event handler is called when the Restart Manager is enabled and Windows Installer 4.0 sends an INSTALLMESSAGE_RMFILESINUSE message to the installation.

For more details, see:

- SdRMFilesInUse
- OnRMFilesInUse

Ability to Specify Additional Compiler Include Paths on a Per-Project Basis

The Compile/Link tab on the Settings dialog box, which is available from the Build menu in InstallShield, now has an Include Paths box. Use this Include Paths box to specify which directories InstallShield should search for source files that have been included in the main installation's InstallScript code through #include statements. It corresponds with the `-i` option for **Compile.exe**, the command-line compiler.

To learn more, see Compile/Link Tab.

Enhancements for .NET Framework 3.0 Support

A new FOLDER_DOTNET_30 InstallScript variable is available. This variable stores the path of the .NET Framework 3.0. The corresponding text substitution for this variable is <FOLDER_DOTNET_30>.

The .NET Framework 3.0 writes the value *InstallSuccess* with value data of 1 to the registry to indicate that it is installed. To check for this InstallSuccess value, you can now use the **Is** function and pass the DOTNETFRAMEWORKINSTALLED constant. You can still pass this constant through the **Is** function to check for the value of *Install*, which is used by earlier versions of the .NET Framework. For more information, see Is.

Two new constants are available for use with the **Is** function:

- REGDB_KEYPATH_DOTNET_30
- REGDB_VALUENAME_INSTALLSUCCESS

New Is Constant for Checking for the Presence of a Minimum Service Pack Number of the .NET Framework

Use the new DOTNETSERVICEPACKINSTALLED constant with the **Is** function to determine whether a particular service pack—or a later version of the service pack—of the .NET Framework is installed. For more information, see Is.

New and Updated Functions for SQL Support

Several InstallScript functions for SQL support have been added or updated.

The following InstallScript functions are now available for InstallScript and InstallScript MSI projects:

- SQLRTInitialize2—Loads the **SQLRT.dll** file for InstallScript projects and the **ISSQLSRV.dll** file for InstallScript MSI projects, and it uses the settings file to initialize the .dll file. This function supersedes the SQLRTInitialize function.

- SQLServerSelectLogin2—Creates a login dialog that lets the targeted end user specify which SQL Server should be used for the current connection, as well as which login credential should be used. This dialog also optionally shows the connection name that is associated with the connection information. In addition, it optionally allows the end user to specify which database catalog should be used for the current connection. This function supersedes the SQLServerSelectLogin function.
- SQLDatabaseBrowse—Creates a dialog that lets the end user display a list of all database catalogs available on the specified database server.
- SQLBrowse2—Creates a dialog that lets an end user display a list of all database servers that are available on the network for the database technologies specified for a connection. This function supersedes the SQLBrowse function.
- SQLRTPutConnectionInfo2—Sets the connection information (the default server, default database catalog, default user name, and default password). This function supersedes the SQLRTPutConnectionInfo function.

The following InstallScript functions are now available for InstallScript projects:

- SQLRTGetLastError2—Returns detailed information about the last error encountered by the SQL run time and loads the proper SQL error message.This function supersedes the SQLRTGetLastError function.
- SQLRTGetErrorMessage—Returns the descriptive message of the last error encountered by the SQL run time when a connection is being opened.
- SQLRTGetScriptErrorMessage—Returns the descriptive message of the last error encountered by the SQL run time when a SQL script is executing.

The following InstallScript function is now available for InstallScript MSI projects:

- SQLRTTestConnection2—Establishes a connection.This function supersedes the SQLRTTestConnection function.

The following InstallScript functions, which were previously available only for InstallScript projects, are now also available for InstallScript MSI projects:

- SQLRTGetConnections
- SQLRTGetConnectionInfo
- SQLRTPutConnectionInfo
- SQLRTGetConnectionAuthentication
- SQLRTPutConnectionAuthentication

InstallShield still supports the functions that are superseded by new versions of the functions. However, if you upgrade a project that was created in InstallShield 12 or earlier to InstallShield 2016, InstallShield does not automatically update the existing InstallScript code to use the new functions.

For more information, see:

- Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects
- SQL Functions

New Is Constant for Checking If a User Is in the Administrators Group

Use the new USER_INADMINGROUP constant with the **Is** function to determine whether a user is in the Administrators group, regardless of whether that user is running the installation with a standard access token. For more information, see Is.

Note that passing the USER_ADMINISTRATOR constant through the **Is** function now returns FALSE if the user is in the Administrators group but the group has the SE_GROUP_USE_FOR_DENY_ONLY security identifier (SID) attribute set (that is, the user is running with a standard access token).

New Functions that Enable a .NET Library Loaded Through InstallScript to Be Unloaded Before the Installation Completes

Two new InstallScript functions are available: DotNetCoCreateObject and DotNetUnloadAppDomain.

- The **DotNetCoCreateObject** function calls functions in .NET assemblies without the assembly being registered for COM interoperability. This function lets you specify the .NET application domain in which the .NET assemblies should be loaded and run.

  The **DotNetCoCreateObject** function is similar to the CoCreateObjectDotNet function. The only difference is that with **DotNetCoCreateObject**, you can specify the .NET application domain that should be loaded; the assembly is then run in this domain.

  For **CoCreateObjectDotNet**, the .NET assembly is loaded into the default application domain after the installation completes; thus, the .NET assembly file is locked until the installation has finished.

- The **DotNetUnloadAppDomain** function unloads the specified .NET application domain and releases any assemblies that are currently loaded into the specified application domain.

New RegDBDeleteItem Function

A new InstallScript function called RegDBDeleteItem is now available. The **RegDBDeleteItem** function deletes values under the per application paths key or the application uninstallation key, depending on the value of nItem. For more information, see RegDBDeleteItem.

New GetStatus Function for Objects

A new InstallScript function called **GetStatus** is now available for InstallScript Object projects. The **GetStatus** function retrieves the current status of the object; that is, the current value of Status.Number. For more information, see GetStatus.

Changes to the ListWriteToFileEx Function and Related Constants

The LWTF_OPTION_APPEND_TO_FILE constant has been added to the InstallScript language. You can pass this constant as the nOptions parameter for the **ListWriteToFileEx** function to append a list to an existing file.

In addition, two existing nOptions constants have been renamed:

- LWFT_OPTION_WRITE_AS_ANSI is now LWTF_OPTION_WRITE_AS_ANSI.
- LWFT_OPTION_WRITE_AS_UNICODE is now LWTF_OPTION_WRITE_AS_UNICODE.

To maintain backwards compatibility, the aforementioned two LWFT_* constants are still available, and they are defined the same way as the corresponding new LWTF_* constants.

For more information, see the following:

- ListWriteToFileEx
- LWTF_OPTION_APPEND_TO_FILE
- LWTF_OPTION_WRITE_AS_ANSI
- LWTF_OPTION_WRITE_AS_UNICODE

Expanded InstallScript Text Substitution Functionality

An InstallScript text-substitution association can now be embedded in another text-substitution association; for example, "<MYTEXTSUB1>" can be associated with "My Text Sub 1 Value" and "<MYTEXTSUB2>" with "Text Sub <MYTEXTSUB1> Embedded". Previously, if a local text-substitution association was embedded in a global text-substitution association, the local text substitution was not performed.

What's New in InstallShield 12 SP1

InstallShield 12 supports a beta version of Microsoft Windows Vista. InstallShield 12 SP1 includes changes that offer support for the RTM version of Microsoft Windows Vista.

New Microsoft .NET 3.0 Prerequisite Available

InstallShield now includes a .NET Framework 3.0 setup prerequisite that you can add to Basic MSI and InstallScript MSI projects.

To learn more about setup prerequisites, see Adding InstallShield Prerequisites, Merge Modules, and Objects to Basic MSI and InstallScript MSI Projects.

Ability to Minimize the Number of UAC Prompts by Advertising as Part of Prerequisite Installation

Use the new Advertise If Prerequisites Are Elevated setting in the Releases view to specify whether your .msi file should be advertised and run after the setup prerequisites in the installation have been successfully installed with elevated privileges on Windows Vista machines. The advertisement may allow end users to avoid the User Account Control (UAC) prompt that would otherwise be displayed for an .msi package that requires elevated privileges. For more information, see Specifying Whether a Product Should Be Advertised If Its InstallShield Prerequisites Are Run with Elevated Privileges.

The Advertise If Prerequisites Are Elevated setting is one of several settings in InstallShield that affect whether an installation triggers UAC consent or credential prompts for elevated privileges. Understanding these different settings will help you create the appropriate UAC experience for your installation when end users run it on Windows Vista systems. It will also enable you to try to minimize the number of UAC prompts that are displayed during your installation. To learn about all of these settings, see Minimizing the Number of User Account Control Prompts During Installation.

Enhanced Validation for the Certified for Windows Vista Program

InstallShield includes several new InstallShield internal consistency evaluators (ISICEs) that help you ensure that your installation follows the best practices outlined in the Certified for Windows Vista program. The new evaluators—ISICE11 through ISICE20—check that all .exe files have embedded manifests, that none of the protected registry keys have been modified, that the .dll and .exe files do not use any obsolete API calls, and more. In addition, ISICE02 now verifies that .ocx, .sys, .cpl, .drv, and .scr files are signed. Previously, ISICE02 verified only that .exe files are signed. ISICE06 now checks the destination path of files that are in your installation and that have the same name as a Windows Protected File. If the destination path for the file in your installation does not match the location of the Windows Protected File, ISICE06 now displays a validation warning instead of a validation error.

When you validate your package, InstallShield lists any validation messages on the Tasks tab of the Output window. Now you can double-click a validation message on the Tasks tab to jump to the area of the Direct Editor that corresponds to the validation message. This functionality is available for ISICEs, as well as for standard ICEs. Previously, it was available only for standard ICEs.

For more information, see:

- ISICEs
- Viewing Validation Results

Additional Changes

For a list of issues that are resolved in InstallShield 12 SP1, see the release notes. The release notes are available from the Help menu in InstallShield.

What's New in InstallShield 12

New Features

InstallShield includes the following new features.

Ability to Target Windows Vista Systems

InstallShield enables you to specify that your installation requires Windows Vista. It also lets you build Windows Vista–related conditions for features and components.

Validation for the Certified for Windows Vista Program

To provide the best possible installation experience for end users, ensure that your installation follows the best practices outlined in the Certified for Windows Vista program. InstallShield now includes a validation suite that helps you verify that your installation meets the installation requirements of the Certified for Windows Vista program for Basic MSI and InstallScript MSI projects. If you want to be able to use the Windows Vista Logo artwork, your application's installation must meet the program's requirements.

In addition, InstallShield now lets you customize the list of internal consistency evaluators (ICEs) that should be used for installation package validation and merge module validation. This is helpful if you do not want to pursue logo certification but you still want to ensure that your installation or merge module meets specific best practices that are verified during validation. To configure validation settings: on the Tools menu, click Options and select the Validation tab.

To learn more, see the following:

- Specifying Whether Validation Should Be Performed at Build Time
- Specifying Which ICEs, ISICEs, ISVICEs and ISBPs Should Be Run During Validation
- ISICEs

User Account Control Support

InstallShield includes support for the User Account Control functionality that Microsoft added for Windows Vista. Use the new Require Administrative Privileges setting in the General Information view to specify at a project-wide basis whether administrative privileges are required for an installation. Also, use the Required Execution Level setting in the Releases view to specify the minimum level required by your installation's **Setup.exe** file for running the installation (the setup launcher, any setup prerequisites, and the .msi file) on Windows Vista platforms.

For details, see:

- Entering Summary Information Stream Data

- Specifying the Required Execution Level for Your Setup Launcher on Windows Vista and Later Platforms

Support for Minimizing Reboots Through the Restart Manager Infrastructure

Restarting the system after an installation is inconvenient for end users. One of the Certified for Windows Vista program requirements is that all installations must contain an option that enables end users to automatically close applications and attempt to restart them after the installation is complete.

To support this quality guideline, an MsiRMFilesInUse dialog is available in all Basic MSI projects. The installation displays this dialog if one or more files that needs to be updated are currently in use during the installation. To learn more, see Minimizing Reboots on Windows Vista and Later Systems.

Digital Signature Enhancements

If you specify digital signature information for your installation, InstallShield automatically adds the necessary information to the **MsiDigitalCertificate** and **MsiPatchCertificate** tables. The **MsiPatchCertificate** table contains the information that is needed to enable User Account Control (UAC) patching. This enables you to create patches that can be installed by non-administrators.

In addition, the Build Installation page in the Project Assistant now offers the ability to specify digital signature information for your installation.

For more information, see:

- Preparing Installations for Non-Administrator Patches
- Digital Signing and Security

Support for Documentation about Custom Action Behavior

The intended behavior of each custom action must be documented for the Certified for Windows Vista program. This is especially helpful if system administrators deploy your product to enterprise environments; they sometimes need to know what the custom actions do. InstallShield now has a new Help File Path setting in the Custom Actions view to help you meet this requirement. Use this setting to specify the path of a document that describes the behavior of a custom action that you create for your project. For details, see:

- Documenting the Behavior of Custom Actions
- Guidelines for Creating Custom Actions that Meet Requirements of the Windows Logo Program

This documentation now describes each of the built-in InstallShield custom actions that are added automatically to InstallShield projects to support different functionality. See InstallShield Custom Action Reference.

Setup Prerequisite Enhancements

Many enhancements have been made to the Setup Prerequisite Editor:

- The user interface has been improved; it now has menus that list easy-to-access commands.
- The new Behavior tab lets you specify whether end users may skip the prerequisite installation. You can also specify how the prerequisite installation should proceed if it appears that the target machine does or does not need to be restarted.
- Now you can create prerequisite installation conditions for DWORD registry comparisons. You can also create conditions for 64-bit machines.

For more details, see:

- Behavior Tab
- Allowing End Users to Choose Whether to Install an InstallShield Prerequisite
- Planning for Issues that Could Occur with an InstallShield Prerequisite Installation
- Specifying the Behavior for an InstallShield Prerequisite that Requires a Restart

Redistributing setup prerequisites is now more flexible than ever. You can specify different methods for supplying each individual setup prerequisite in your installation. This enables you to store some of the setup prerequisite files on the source media; compress some of the setup prerequisite files into **Setup.exe**, to be extracted at run time; and download some of the setup prerequisite files. In addition, you can now assign release flags to setup prerequisites. Then you can include and exclude any combinations of setup prerequisites when you build different releases.

To learn more, see:

- Specifying a Run-Time Location for a Specific InstallShield Prerequisite
- Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level
- Assigning Release Flags to InstallShield Prerequisites

Windows Installer 4.0 Log File Support on a Project-Wide Basis

InstallShield provides support for the option to easily log installations run with Windows Installer 4.0 on a project-wide basis without having to use the command line or configure log parameters through the registry. To enable logging, specify Yes in the new Create MSI Logs setting in the General Information view. To override the default logging parameters, edit the **MsiLogging** property in the Property Manager view. If the installation is run on a target system that has Windows Installer 4.0, the installer creates a log file and populates the **MsiLogFileLocation** property with its path. In addition, a **Show the Windows Installer log** check box is added to the SetupCompleteSuccess, SetupCompleteError, and SetupInterrupted dialogs.

For more information, see Specifying Whether Windows Installer Installations Should Be Logged.

Registry Reflection Support for 64-Bit Systems

InstallShield includes support for the new Windows Installer 4.0 attribute called msidbComponentAttributesDisableRegistryReflection in the **Component** table. To easily configure this attribute, set the Disable Registry Reflection setting for a selected component in the Components view. Registry reflection keeps the 32-bit registry view and the 64-bit registry view in sync on a target machine. Only 64-bit systems with Windows Installer 4.0 support this setting. Other systems ignore it. For more specific information, see Enabling and Disabling Registry Reflection.

Multilingual User Interface (MUI) Support

If you are preparing an installation for a multilingual application and Windows Installer 4.0 will be running the installation, you can now use InstallShield to create shortcuts that include support for the Windows multilingual user interface (MUI). Four new settings are available in the Shortcuts view for a selected shortcut:

- Display Resource DLL
- Display Resource ID
- Description Resource DLL
- Description Resource ID

These new settings correspond with the four new columns in the **Shortcut** table for Windows Installer 4.0. To learn more, see Shortcuts View.

Additional InstallShield Collaboration Licenses for Premier Edition

InstallShield Premier Edition now includes a five-pack of licenses for InstallShield Collaboration for Visual Studio. You can use these licenses to install InstallShield Collaboration on development systems that do not have InstallShield but do have Visual Studio .NET 2002, Visual Studio .NET 2003, or Visual Studio 2005.

InstallScript Rearchitecture Enhancements for InstallScript MSI Projects and Basic MSI Projects with InstallScript Custom Actions

The InstallScript MSI architecture has had a number of issues with security (COM/DCOM) and other areas that can cause some installations to fail for various reasons. The architecture has been improved dramatically for InstallShield 12 to resolve these issues and to make InstallScript MSI a more reliable project type. The improvements also help increase the reliability of Basic MSI projects that include InstallScript custom actions.

To learn more, see the following:

- Run-Time Behavior for Basic MSI Installations with InstallScript Custom Actions
- Run-Time Behavior for InstallScript MSI Installations
- Upgrading Projects from InstallShield 11.5 or Earlier

DIFx 2.01 Support

InstallShield includes support for the latest version of Driver Install Frameworks for Applications (DIFx). This new version, which includes the latest binary files from Microsoft, is available for any Basic MSI, InstallScript, or InstallScript MSI projects that you create in InstallShield.

Enhancements for Setup.exe and Update.exe Command-Line Parameters

Some of the command-line parameters for **Setup.exe** and **Update.exe** were added or modified:

- `/clone_wait` (InstallScript only)
- `/runfromtemp` (Basic MSI, InstallScript, and InstallScript MSI)
- `/v"ISSCRIPTCMDLINE=\"<Option1> <Option2>\""` (Basic MSI only)
- /hide_progress (Basic MSI, InstallScript, and InstallScript MSI projects)
- /hide_splash (InstallScript and InstallScript MSI projects)

For more information, see Setup.exe and Update.exe Command-Line Parameters.

Registry and File Filtering Enhancements for COM Extraction and Dependency Scanners

To prevent InstallShield from extracting undesired COM data from a COM server, you can edit a new **Filters.xml** file that is installed with InstallShield. Editing this **Filters.xml** file enables you to customize the list of registry keys that will be excluded from COM extraction.

The **Filters.xml** file also now lists files that the Static, Dynamic, and Visual Basic dependency scanners will exclude or include. Previously, two different files—**Userscan.ini** and **Iswiscan.ini**—were used to list excluded and included files.

For more information, see:

- Filtering Registry Changes for COM Extraction
- Filtering Files in Dependency Scanners

Internet Explorer 7.0 Compatibility

Several areas of InstallShield have been revised so that it is now compatible with Internet Explorer 7.0. The Transform Wizard, Custom Action Wizard, the String Table Editor, the Distribute view, and the General Information view are all examples of areas that have been updated.

Enhanced Start Page

The list of recently opened projects that are displayed on the Start Page now includes a column that shows the project type. In addition, the maximum number of projects that are listed has been increased from four to eight.

InstallScript Enhancements and New Functionality

Several enhancements have been made to the InstallScript language. New and revised variables, functions, and constants are now available.

System Variable Changes and Text Substitution Additions

A system variable called DISK1SETUPEXENAME has been added.

Two system variables have new default values:

- UNINST
- UNINSTALL_STRING

Two new InstallScript text substitutions are available:

- DISK1SETUPEXENAME
- SELECTED_LANGUAGE

For more information, see System Variables.

Script Variable Additions

Two new script variables are available:

- INSTALLSCRIPTMSI
- BASICMSI

Function and Predefined Constant Changes

A new LANGUAGE_SUPPORTED predefined constant is available for use with the **Is** function. To learn more, see Is.

A new ISOSL_WINVISTA predefined constant is available for use with the **FeatureFilterOS** function and the SYSINFO structure variable. For more information, see FeatureFilterOS function and the SYSINFO.

The **DoInstall** function is now similar to the **LaunchAppAndWait** function. For more details, see DoInstall.

Upgrade Details

For InstallShield 12, a number of improvements were made to the InstallScript MSI architecture to resolve issues with security (COM/DCOM) and other areas that can cause some installations to fail for various reasons. The architecture was improved for InstallShield 12 to resolve these issues and to make InstallScript MSI a more reliable project type. The improvements also help increase the reliability of InstallScript projects, as well as Basic MSI projects that include InstallScript custom actions.

In addition, the merge module project type has been enhanced: it now has the same InstallScript view that is available in Basic MSI projects. This enhanced version of the merge module project type replaces the InstallScript MSI object project type, which is no longer available in InstallShield.

To learn more, see Upgrading Projects from InstallShield 11.5 or Earlier.

Target System Requirements

You can use InstallShield to rapidly build, test, and deploy installations that target Windows-based systems.

Requirements For Desktop Computers

Operating System



Project • *For all project types except for Suites (Advanced UI, and Suite/Advanced UI project types), Windows XP SP3 and Windows Server 2003 SP2 are the minimum versions of Windows that are required for target systems that run the installations that are created in InstallShield. For Suites, Windows Vista and Windows Server 2008 are the minimum versions of Windows that are required for target systems.*

Target systems must meet the following minimum operating system requirement:

- Windows XP SP3
- Windows Server 2003 SP2
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

Target systems must also support the SSE2 instruction set.

Installer Engine Requirements

The minimum target system requirements for the various installer engines are as follows.

Table 1-3 • Target System Requirements for Desktop Computers

Installer Engine	Operating System and Other Requirements
Windows Installer 5.0 **(This is not available as a redistributable.)**	Windows 7 or later, Windows Server 2008 R2 or later
Windows Installer 4.5	Windows XP SP2 or later, Windows Server 2003 SP1 or later
Windows Installer 4.0 **(This is not available as a redistributable.)**	Windows Vista, Windows Server 2008
Windows Installer 3.1	Windows 2000 SP3 or later, Windows Server 2003 or later
Windows Installer 3.0	Windows 2000 SP3 or later, Windows Server 2003 or later
Windows Installer 2.0	Windows 95 or later (Note that InstallShield does not include support for Windows 95, Windows 98, Windows NT 4, or Windows Me.)
Suite/Advanced UI	Windows Vista or later, and Windows Server 2008 or later
Advanced UI	Windows Vista or later, and Windows Server 2008 or later
InstallScript	Windows XP SP3 or later, Windows Server 2003 SP2 or later

For more information about Windows Installer redistributables, see Adding Windows Installer Redistributables to Projects.

Targeting 64-Bit Operating Systems

Microsoft designed 64-bit versions of Windows to allow existing 32-bit applications to continue to work seamlessly. They also designed 64-bit versions of Windows in such a way to allow a recompiled version of the same code to work seamlessly as a 64-bit application. To provide this support, 64-bit versions of Windows isolate the 32-bit and 64-bit from each other in two main ways: their files are stored in separate locations (Program Files vs. Program Files (x86); System32 vs. SysWow64), and their registry keys are separated (HKLM\Software vs. HKLM\Software\Wow6432Node).

In some cases, such as installation, what is normally a beneficial separation becomes a challenge. Typically installations are 32-bit applications themselves (in order to run on 32-bit machines), and accessing 64-bit locations to install a 64-bit application is more complex than a standard file copy or registry write. Each project type that is available in InstallShield handles this complexity in different ways.

Windows Installer–based installations disallow access to 64-bit locations except when the installation targets only 64-bit systems; however, they support running 64-bit DLL code whenever running on a 64-bit system. Note that some target systems—such as Windows Server Core systems—may not have 32-bit Windows-on-Windows (WOW64) support. InstallShield lets you create pure 64-bit installations that work on those systems. For more information, see Targeting 64-Bit Operating Systems with Basic MSI and InstallScript MSI Installations.

InstallScript installations are always 32 bit so they can run only 32-bit DLL code (or launch 32-bit or 64-bit .exe files). However, all InstallScript installations support accessing 64-bit locations through the 64-bit component setting, as well as through other methods. For additional details, see Targeting 64-Bit Operating Systems with InstallScript Installations.

Suite/Advanced UI and Advanced UI installations enable you to reference either 32-bit or 64-bit locations, and they can launch 32-bit or 64-bit installations, depending on the target system. However, all extension DLL code in a Suite/Advanced UI or Advanced UI project must be 32 bit. To learn more, see Targeting 64-Bit Operating Systems with Suite/Advanced UI and Advanced UI Installations.

Targeting 64-Bit Operating Systems with Basic MSI and InstallScript MSI Installations



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Note that if you create an InstallScript MSI project and change the Template Summary property from Intel to x64, InstallShield creates a 64-bit MSI installation combined with a 32-bit InstallScript MSI installation at build time.

Also note that InstallScript MSI projects do not have support for architecture validation.

If you are writing applications for 64-bit Windows-based systems, you will need a way to install your 64-bit files and other data. Although 32-bit Windows Installer–based installations can run on most 64-bit machines, they cannot install to 64-bit locations. The Windows Installer service provides support for 64-bit installations. These installations can be designed and built on 32-bit machines, but they can run only on 64-bit machines. Through the use of release flags, you can build two installations (one 32 bit and one 64 bit) from a single project.

Configuring the Template Summary Property

The Template Summary property determines whether an .msi package is a 32-bit package or a 64-bit package.

If your installation targets 64-bit systems, configure the Template Summary property with the appropriate 64-bit value (x64 or Intel64). This allows end users to install your product in 64-bit locations on 64-bit systems; it also prevents end users from installing your product on 32-bit systems.

You can use the Template Summary setting in the General Information view to configure the Template Summary property. You can also set the Template Summary setting for a product configuration in the Releases view. Any value entered in the Releases view overrides the value set in the General Information view.

To learn more, see Using the Template Summary Property.

Creating 64-Bit Components

To specify that a component is 64 bit, select Yes for the component's 64-Bit Component setting. If you build a release that has at least one 64-bit component and the Template Summary property is set to a 64-bit value, the result is a 64-bit package.

For more information about the 64-Bit Component setting, as well as additional component settings, see Component Settings.

Extracting COM Data from 64-Bit COM Servers

If you are using InstallShield on a 64-bit operating system, InstallShield can extract COM data from a 64-bit COM server. In order to install the data to the correct locations, the component must be marked as 64 bit.

Note that if you use InstallShield on a 32-bit system, 64-bit COM extraction is not available.

To learn about extracting COM data, see Extracting COM Information from a COM Server.

Self-Registering 64-Bit COM Servers

InstallShield supports 64-bit self-registration of COM servers. If you mark a component as 64 bit and then add a file to that component, you can select the file's Self Register check box to enable 64-bit self-registration of that file during installation. The Self Register check box is on the file's Properties dialog box.

InstallShield also supports 64-bit self-registration of dynamically linked COM servers. To enable this, select the **Self-Register all files** check box on the Dynamic File Link Settings dialog box when you add the files dynamically to a 64-bit component. For more information, see Adding Dynamic File Links to Components.

Enabling or Disabling Registry Reflection

InstallShield includes support for the Windows Installer 4.0 and later attribute msidbComponentAttributesDisableRegistryReflection in the **Component** table. To use this attribute, set the Disable Registry Reflection setting for a selected component in the Components view to Yes. Registry reflection keeps the 32-bit registry view and the 64-bit registry view in sync on a target machine. Only 64-bit systems with Windows Installer 4.0 and later support this setting. Other systems ignore it. For more specific information, see Enabling and Disabling Registry Reflection.

Adding Other 64-Bit Elements

InstallShield includes support for additional 64-bit elements. For example, InstallShield includes 64-bit .NET Framework redistributables that you can add to your project. Custom actions can include 64-bit JScript or VBScript code, and 64-bit folder properties give you access to essential operating system folders such as the 64-bit Program Files folder. In addition, you can use the Setup Prerequisite Editor to configure setup prerequisites that have conditions that check for 64-bit requirements on the target system.



Note • *64-bit support requires Windows Installer version 2.0 or later.*

Specifying the Appropriate Architecture Validation

Basic MSI projects include a product configuration setting that lets you specify whether you want to use architecture validation at build time. Architecture validation enables you to detect potentially problematic cases in which your installation may try to install product files or use run-time binaries that may not match the architecture of a target system.

For example, if end users may run your installation on 64-bit Windows Server Core systems that do not have 32-bit Windows-on-Windows (WOW64) support, architecture validation can help you identify any 32-bit product files or 32-bit custom action files in your installation; 32-bit binaries cannot be loaded on 64-bit target systems without WOW64 support.

Architecture validation also enables you to generate pure 32-bit .msi packages that contain only 32-bit versions of the built-in InstallShield custom action DLLs, or pure 64-bit .msi packages that contain only 64-bit versions of the built-in InstallShield custom action DLLs.

To specify what type of architecture validation you want to use at build time and identify whether you want to build a pure 32-bit or 64-bit package, use the Architecture Validation setting for a product configuration in the Releases view.

For more details, see Selecting the Appropriate Type of Architecture Validation for Builds.

Tips for Building Two Installations—One 32 Bit and One 64 Bit—from a Single Project

If you want to be able to build two installations (one 32 bit and one 64 bit) from a single project, consider using release flags. You can assign release flags to features, InstallShield prerequisites, and chained .msi packages, as necessary, to differentiate the 32-bit items from the 64-bit items. Then you can configure each release or product configuration in the Releases view to include or exclude certain items that have a particular release flag at build time. You can also conditionally launch certain custom actions based on release flags, so that 32-bit custom actions are launched during a 32-bit installation and 64-bit custom actions are launched during a 64-bit installation.

To learn more, see:

- Release Flags
- Filtering Based on Release Flags
- Conditionally Launching Custom Actions Based on Release Flags

InstallShield lets you override the values of your project's path variables for each release in your project. This functionality enables you to essentially replace certain files and folders in your project with others at build time, depending on the particular release that you are building.

For example, you might use this functionality to swap out the binaries for custom actions. If you have set up separate releases for 32-bit and 64-bit target systems, you can override the path variable that is used to refer to the DLL that is selected for a custom action. Then InstallShield could include a 32-bit DLL for your 32-bit release and a 64-bit DLL for you 64-bit release. Note that overriding path variables to swap out files that your installation is installing is not recommended. This is because you should use separate components for 32-bit and 64-bit versions of a file.

To override one or more path variables in your project, use the Path Variables Overrides setting, which is on the Build tab for a release in the Releases view. For more information, see Build Tab for a Release.

Targeting 64-Bit Operating Systems with InstallScript Installations



Project • *This information applies to InstallScript projects.*

With a single InstallScript project, you can create one installation that installs to 32-bit locations on 32-bit systems, and to 64-bit and 32-bit locations as needed on 64-bit systems.

InstallScript installations are always 32-bit so they can run only 32-bit DLL code (or launch 32-bit or 64-bit .exe files). However, all InstallScript installations support accessing 64-bit locations through the 64-bit component setting, as well as through other methods.

Reading from and Writing to the 64-Bit System32 Folder

InstallScript includes the following system variables that let you choose between 32-bit and 64-bit System32 folders on target systems:

- WINSYSDIR
- WINSYSDIR64

Although the WINSYSDIR64 variable is set to the 64-bit Windows folder, 64-bit Windows includes functionality to automatically redirect 32-bit applications (such as the InstallScript engine) to the 32-bit System32 folder (SysWOW64). Therefore, if you are installing files or folders to WINSYSDIR64, you can disable file system redirection by placing the folders or files in a component that is marked as 64 bit. To specify that a component is 64 bit, select Yes for the component's 64-Bit Component setting. For more information about the 64-Bit Component setting, as well as additional component settings, see Component Settings.

As an alternative, you can use the constant WOW64FSREDIRECTION with the functions Disable and Enable to disable 64-bit Windows file system redirection while writing to or reading from the WINSYSDIR64 folder through your InstallScript code, and then re-enable it once the write or read is complete. Since some Windows functionality that could be used by the installation requires that redirection be enabled to work, Windows documentation recommends that you disable redirection only for as long as necessary.

Reading from and Writing to Other 64-Bit Destination Folders

InstallScript includes several addition system variables that let you choose between other 32-bit and 64-bit locations on target systems:

- COMMONFILES
- COMMONFILES64
- FOLDER_APPLICATIONS
- FOLDER_APPLICATIONS64
- PROGRAMFILES
- PROGRAMFILES64

Reading from and Writing to 64-Bit Registry Locations

The 64-bit version of Windows includes functionality that maps some 64-bit registry locations to 32-bit equivalents. For example, on a 64-bit version of Windows, HKEY_LOCAL_MACHINE\Software\Wow6432Node is the 32-bit equivalent of HKEY_LOCAL_MACHINE\Software.

If you want to install registry data to a 64-bit area of the registry without having it redirected to a 32-bit area, you can place the registry data in a component that is marked as 64 bit. To specify that a component is 64 bit, select Yes for the component's 64-Bit Component setting. For more information about the 64-Bit Component setting, as well as additional component settings, see Component Settings.

As an alternative, you can use the REGDB_OPTION_WOW64_64KEY option with the REGDB_OPTIONS variable to avoid registry redirection while editing the registry or reading from the registry through your InstallScript code, and then use the REGDB_OPTION_USE_DEFAULT_OPTIONS option with REGDB_OPTIONS to enable changes to the 32-bit area of the registry. Since some Windows functionality that could be used by the installation requires that redirection be enabled to work, Windows documentation recommends that you disable redirection only for as long as necessary.

Installing Self-Registering 64-Bit COM Servers

InstallShield supports 64-bit self-registration of COM servers. For the component that contains the self-registering DLL, .exe, .tlb, or .olb file, select Yes for the component's Self-Register setting.

Note that 32-bit self-registering files should not be installed to the 64-bit System32 folder, since file system redirection cannot be disabled for self-registration. Thus, if you have a 32-bit self-registering COM server that needs to be installed to the 32-bit System32 folder, ensure that No is selected for the 64-Bit Component setting of the component that contains the COM server.

About Add or Remove Programs Information

InstallScript installations do not support reading or writing Add or Remove Programs information in the 64-bit part of the registry. Therefore, InstallScript installations always install Add or Remove Programs information in the 32-bit part of the registry. In addition, the REGDB_OPTION_WOW64_64KEY option does not disable registry reflection for registry functions such as **CreateInstallationInfo**, **MaintenanceStart**, **RegDBGetItem**, **RegDBSetItem**, **RegDBGetAppInfo**, **RegDBSetAppInfo**, and **RegGetUninstCmdLine**.

Targeting 64-Bit Operating Systems with Suite/Advanced UI and Advanced UI Installations

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Note that Suite/Advanced UI and Advanced UI installations are always 32-bit. Therefore, they can run 32-bit DLL code, but they cannot run 64-bit DLL code.

Suite/Advanced UI and Advanced UI installations enable you to reference either 32-bit or 64-bit locations, and they can launch 32-bit or 64-bit installations, depending on the target system.

For example, if you include in your Suite/Advanced UI project one installation that should target 32-bit systems, and another installation that should target 64-bit systems, you could create an eligibility condition in the Packages view for both of these packages; for the 32-bit package, the eligibility condition would contain a platform condition that checks for the x86 architecture, and for the 64-bit package the eligibility condition would contain a platform condition that checks for the x64 or IA64 architecture. For more information, see Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects.

Launching InstallShield with vs. Without Administrative Privileges

If you launch InstallShield without administrative privileges, the following functionality is not available:

- **COM extraction**—Extracting COM information from a COM server requires administrative privileges.

  If you specify that you want to have COM information extracted from a COM server in your project, and then you try to build a release while running InstallShield without administrative privileges, build error -6017 occurs.

- **Redistributable downloading**—Downloading redistributables from within the Redistributables view requires administrative privileges. This is because InstallShield tries to download the files to a per-machine location, which requires administrative privileges.

  If you try to download a redistributable from within the Redistributables view but you do not have administrative privileges, InstallShield displays the following message:

  ```
  The download failed; make sure you are running as Administrator, and that your machine is connected
      to the Internet. Would you like to try again?
  ```

- **Ability to specify All Users locations for InstallShield prerequisites**—The Prerequisites tab on the Options dialog box is where you specify the folders that contain the InstallShield prerequisites that should be displayed in the Redistributables view and the Prerequisites view. Modifying the All Users location on that tab requires administrative privileges, since InstallShield writes the information to a per-machine location in the registry. Therefore, the All Users location on that tab is disabled if you are running InstallShield without administrative privileges.

- **Ability to specify All Users locations for merge modules**—The Merge Module tab on the Options dialog box is where you specify the folders that contain the merge modules that should be displayed in the Redistributables view. Modifying the All Users location on that tab requires administrative privileges, since InstallShield writes the information to a per-machine location in the registry. Therefore, the All Users location on that tab is disabled if you are running InstallShield without administrative privileges.

- **Ability to edit the locations for Regasm.exe and InstallUtilLib.dll**—The .NET tab on the Options dialog box is where you specify the locations of **Regasm.exe** and **InstallUtilLib.dll** files, which are utilities that are included with the .NET Framework. These utilities are used for COM interop and .NET custom actions. Modifying these locations on the .NET tab requires administrative privileges, since InstallShield writes the information to a per-machine location in the registry. Therefore, these location settings on that tab are disabled if you are running InstallShield without administrative privileges.

- **Ability to specify the frequency for checking for InstallShield updates**—The **Check for software updates** option on the Updates tab of the Options dialog box is where you specify how often you want InstallShield to check for software updates. Modifying the frequency on that tab requires administrative privileges, since InstallShield writes the information to a per-machine location in the registry. Therefore, the **Check for software updates** option on that tab is disabled if you are running InstallShield without administrative privileges.

If you switch between full Administrator and non-Administrator contexts and you use mapped-drive locations in your projects, you may encounter issues. For example, if you map a drive letter to a shared network folder through Windows Explorer without administrative privileges, you can access this drive letter in a non-administrative instance of InstallShield, but not in an administrative instance. Likewise, if you map a drive letter to a shared network folder through Windows Explorer with administrative privileges, you can access this location in an administrative instance of InstallShield, but not in a non-administrative instance. Thus, if you want to reference network locations in your project, consider using UNC paths (such as \\server\share) or mapping drive letters both with and without administrative privileges.

Note that if you are using InstallShield from within Visual Studio, you may not have administrative privileges. By default, if you launch Visual Studio by double-clicking its shortcut on a Windows Vista or later system, you will not have administrative privileges.

Task ***To run InstallShield from within Visual Studio with administrative privileges on a Windows Vista or later system:***

1. On the Start Menu, right-click the Visual Studio shortcut and then click **Run as administrator**.
2. Create a new InstallShield project or open an existing one. For more information, see one of the following:
   - Creating InstallShield Projects in Microsoft Visual Studio
   - Opening InstallShield Projects in Microsoft Visual Studio

Developing and Building Installations on 32-Bit vs. 64-Bit Systems

InstallShield is a 32-bit application that runs on both 32-bit and 64-bit systems. In some cases, InstallShield behaves differently, depending on whether you are using it on a 32-bit system or a 64-bit system; you may also encounter differences depending on which operating system you are using. The following sections explain these differences.

Note that like InstallShield, the command-line build (**ISCmdBld.exe**), the Standalone Build, and the automation interface are also 32-bit applications. Therefore, the same platform-specific differences occur for these tools.

Loading the Automation Interface

The automation interface is a 32-bit interface; therefore, it must be loaded from a 32-bit process. If you are using the automation interface on a 64-bit machine, you may need to load the automation interface through a 32-bit executable file. Otherwise, you may encounter errors.

For more information, see Using the Automation Interface on a 64-Bit System.

Adding System Files to Your Project

When you are using InstallShield on a 64-bit system and you want to add to your project a system file whose source location is the 64-bit System folder (System32) on your development machine, it is not possible to drag the file from one of the source computer panes at the top of the Files and Folders view to the appropriate location in one of the destination computer panes.

To add a 64-bit System32 file on your development machine to your InstallShield project, you can browse to the Sysnative folder on your machine and then select the appropriate file for your project. To learn more, see Adding Files from Your 64-Bit Source Machine's 64-Bit System32 Folder.

Viewing Both the 32- and 64-Bit Areas of the Source Machine's Registry on 64-Bit Development Systems

If you are using InstallShield on a 64-bit development system, the Registry view in InstallShield displays both the 32-bit and 64-bit areas of your machine's registry:

- HKEY_LOCAL_MACHINE\Software
- HKEY_LOCAL_MACHINE\Software\Wow6432Node

This support enables you to drag and drop entries from those source areas to the appropriate areas in the destination pane of this view when you are configuring registry data changes for your project.

Note that if you want your installation to install registry data to a 64-bit area of the registry on 64-bit target systems without having it redirected to a 32-bit area, you must place the registry data in a component that is marked as 64 bit. Simply dragging 64-bit data from the source panes in the Registry view to a location in one of the destination panes of the view does not mark the component as 64 bit. For more information, see the following:

- Targeting 64-Bit Operating Systems
- Dragging and Dropping Registry Entries to Create Registry Keys

Extracting COM Data from 64-Bit COM Servers

If you want to extract COM data from a 64-bit COM server at design time or at build time, you must be using InstallShield on a 64-bit operating system. If you use InstallShield on a 32-bit system, 64-bit COM extraction is not available.

To learn about extracting COM data, see Extracting COM Registration Data at Build Time.

Scanning 64-Bit Files for Dependencies

If you use the dependency scanners in InstallShield to help you identify dependencies that you may need to add to your project, these scanners can check 64-bit files in your project for dependencies only if you are using InstallShield on a 64-bit operating system.

If you are using InstallShield on a 32-bit system, the dependency scanners in InstallShield cannot check 64-bit files in your project for dependencies. To add 64-bit dependencies to a project on a 32-bit system, consider manually adding the required files and merge modules to your project.

To learn about scanning files for dependencies, see Identifying Application Dependencies.

Scanning 64-Bit .NET Assemblies for Dependencies and Properties

If you use InstallShield on a 64-bit version of Windows Vista or later or a 64-bit version of Windows Server 2008 or later, and you use one of the built-in methods for detecting dependencies, InstallShield can scan for 64-bit dependencies of the 64-bit .NET assemblies in your project.

These methods also scan for 32-bit dependencies of the 32-bit .NET assemblies in your project.

Note that if you use InstallShield on a 32-bit version of Windows, these built-in scans can check for only 32-bit dependencies of the 32-bit files in your project. If your project includes 64-bit files, you can manually add any dependencies to the project as needed.

To learn more, see the following:

- Identifying Properties and Dependencies of .NET Assemblies
- Scanning 64-Bit .NET Assemblies for Dependencies

Using 64-Bit .NET Installer Classes and COM Interop

If you are using InstallShield on a 64-bit version of Windows, the .NET tab on the Options dialog box—which is displayed when you click Options on the Tools menu in InstallShield—lets you specify different paths for the 32-bit and 64-bit locations of the **Regasm.exe** and **InstallUtilLib.dll** files that are included with the .NET Framework. InstallShield uses the paths that you specify at build time for releases that include .NET installer classes and COM interop.

If you are building from the command line with **ISCmdBld.exe** on a 64-bit version of Windows, and you use the existing `-t` parameter to specify the path of the 32-bit version of the .NET Framework, **ISCmdBld.exe** uses the 64-bit location of `Regasm.exe` and **InstallUtilLib.dll** for 64-bit .NET installer classes and COM interop.

If you are building through MSBuild or Team Foundation Server (TFS) and you use the existing DotNetUtilPath parameter on the InstallShield task to specify the path of the 32-bit version of the .NET Framework, the build uses the 64-bit location of **Regasm.exe** and **InstallUtilLib.dll** for 64-bit .NET installer cases and COM interop.

Note that if you are using InstallShield on a 32-bit operating system, the setting for the 64-bit location is disabled, and only 32-bit support for .NET installer cases and COM interop is available. This support also applies to building from the command line with **ISCmdBld.exe**, and building through MSBuild or TFS.

To learn more, see:

- .NET Tab (on the Options dialog box)
- ISCmdBld.exe
- Microsoft Build Engine (MSBuild)

Using Help

Flexera Software understands the importance of having useful information and help resources at your fingertips. In addition to the inline help embedded within various views of the InstallShield interface, InstallShield includes a help library that is installed with the product, an online help system that is available on the Web, and documentation that is available in PDF format.

InstallShield Help Library

When you have questions about your product, first consult the InstallShield Help Library. It is a complete user guide.

You can access the InstallShield Help Library from the **Help** menu in InstallShield, by pressing F1, or by clicking Help buttons in the interface.

No Internet connectivity is required to view the InstallShield Help Library. Essentially, the online help viewer is a tool that you can use to display, search, and filter technical information based on your personal needs.

Using Context-Sensitive Help

While working on a project, clicking a software object displays its help information in the Help window. This is also called "context-sensitive" help.

Help information is also available for each of the properties of a selected software object displayed in the Explorer window, which provides instructions for setting that property.

Web-Based Online Help

Web-based online help is available to you 24 hours a day, seven days a week, on our Web site at http://helpnet.fleerasoftwware.com. This help resource center provides near real time updates to documentation.

Documentation as PDF

InstallShield documentation is also available as a PDF at: https://flexeracommunity.force.com/customer/CCDocumentation.

Contacting Us

Flexera Software is headquartered in Itasca, Illinois, and has offices worldwide.

To contact us or to learn more about Flexera Software and its products, visit the Flexera Software website at: http://www.flexerasoftware.com.

2

Getting Started

InstallShield provides powerful features and time-saving tools that make authoring reliable Windows Installer and InstallScript installations easy. The InstallShield Help Library is a resource that will help you harness the full potential of InstallShield. The help topics you might want to read first depend upon your previous experience with InstallShield installation-authoring software. The table below directs you to various topics based on your level of experience.

Table 2-1 • Getting Started Road Map

Level of Familiarity	Top Help Topics
You have not created installations previously.	To learn general information about installations, refer to the following topics: • Installation Fundamentals • Application Lifecycle
You are new to InstallShield.	If you have created installations previously but you do not have experience using InstallShield, see the following topics: • Working with the InstallShield Interface • Working with Projects • Basic MSI Project Tutorial • Globalization Tutorial • InstallScript Project Tutorial • Run-Time Language Support in InstallShield • Supported Application Programming Languages

Table 2-1 • Getting Started Road Map (cont.)

Level of Familiarity	Top Help Topics
You have used other Flexera Software products.	If you have used other products from Flexera Software, refer to any of the following topics: • Upgrading from Earlier InstallShield Versions • Upgrading from Other InstallShield Editions • Project Assistant • Installation Projects Overview
You are an intermediate-level or advanced-level user of InstallShield.	If you are familiar with InstallShield, you may want to review the following topics: • Command-Line Tools • Errors and Warnings • InstallScript Language Reference • Running Installations in Silent Mode • Configuring Advanced Settings for InstallShield

Installation Fundamentals

An installation, in its simplest terms, is the "package" used to install your files and programs onto your user's machine. It is a complete collection of the application files, as well as logic that interacts with the installer engine. The primary task of any installation is to transfer the application files from the source medium to the end user's computer. The complexities of the Windows operating system make it anything but simple to create an effective, coherent installation without the aid of a utility such as InstallShield.

An installation is divided into three levels: products, features, and components. The following diagram illustrates this hierarchy:

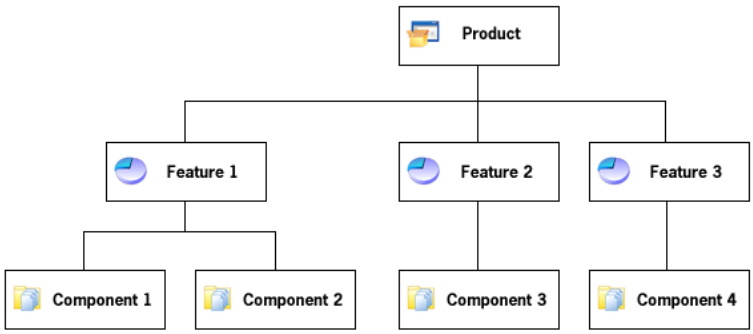


Figure 2-1: Installation Hierarchy

A *product* is the highest level of organization in an installation project. A product is usually one main application (for example, a word processor) and all of the files and data that the application requires; a suite of applications may also be a product.

A *feature* is the smallest installable part of a product, from the end user's perspective. As the designer of an installation program, you usually allow the user to choose which features to install and which features to leave on the source media. In a word processor product, the main executable may be one feature, and optional dictionaries may be a separate feature. A feature should be self-contained, in the sense that a feature should not require sibling features. For example, a thesaurus feature should not require a dictionary feature that the user can choose not to install. However, you can design features to contain subfeatures, which allow the end user finer control over which files and data to install.

Each feature in a project is made up of one or more *components*. A component is the smallest installable part of a product from the installation developer's perspective; components are invisible to the end user. Each component contains files (and other resources) with similar properties. For example, all of the files in a component will be installed in the same directory on an end user's machine, and all of the files in a component should apply to the same operating system or language. A dictionary feature might contain several language-specific dictionary components. In addition to containing files, components generally contain registry data, shortcuts, file extension information, and other system data to write to a user's machine.

When you are designing an installation, the overall task is to separate the product's files into components based on the files' destination, operating system, language, and other properties, and to associate each component with one or more features.

Application Lifecycle

Your application lifecycle should not end when your customer installs your application. As a software vendor, the success of your application goes well beyond the initial installation on the customer's desktop. Customers expect to have easy access to product updates, enhancements, and critical information. Your ability to communicate with your customers and monitor the health of your application is vital to your ongoing growth and profitability.

Too often, software vendors require their customers to initiate communication. Vendors who are not proactively creating an ongoing dialog are missing a tremendous opportunity. Unless the customer is an active visitor to your Web site or public user communities, they miss important information about updates, upgrades, hot fixes, and general technical bulletins. You miss revenue and service opportunities.

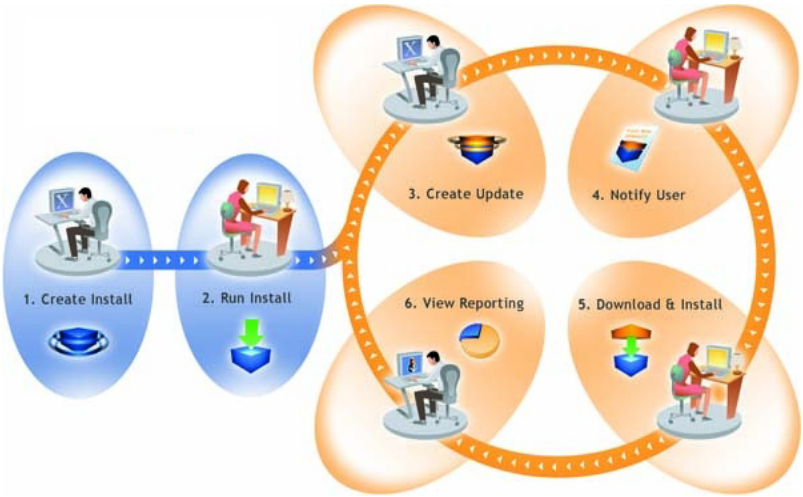


Figure 2-2: How FlexNet Connect Manages the Application Lifecycle

The above diagram illustrates how FlexNet Connect is used to manage the application lifecycle:

1. **Create Install—**InstallShield makes it easy for software developers to create installations that run on any platform.
2. **Run Install—**Installations created with InstallShield technology have successfully installed on over 400 million machines worldwide.
3. **Create Update—**InstallShield enables software developers to rapidly build patches and updates.
4. **Notify User—**FlexNet Connect notifies every user that a new update is ready to be installed.
5. **Download & Install—**FlexNet Connect downloads the update and installs it in one seamless, integrated process.
6. **View Reporting—**FlexNet Connect provides immediate feedback on the update's adoption rate.

Starting InstallShield

Each time you open InstallShield (with the exception of the initial launch after installing the product), you are taken directly to the InstallShield Start Page. The Start Page provides quick access to product information, to recently opened projects, and to InstallShield resources.

InstallShield Start Page

The InstallShield Start Page provides quick access to product information, to recently opened projects, and to InstallShield resources. The Start Page includes the following sections:

Table 2-2 • Sections on the Start Page

Section	Description
Project Tasks	Click a project task to quickly create a new project, open an existing project, or browse to one of the sample projects included with the InstallShield installation.
Help Topics	Frequently accessed help topics are listed in this section. To access the entire InstallShield Help Library from the Start Page, press F1 or click the Help Library link in the Resources section.
(Recently Opened Projects)	The section in the middle of the Start Page lists your most recently accessed projects, the project types, and the dates on which they were last modified.
Getting Started	Click the Getting Started heading for guidance on what areas of the InstallShield Help Library to read, based on your level of experience with InstallShield and installation-authoring tools.

Table 2-2 • Sections on the Start Page (cont.)

Section	Description
Resources	The Resources section contains a number of links to connect you to helpful InstallShield information. • **Help Library**—Displays the InstallShield documentation. • **InstallShield Community**—Provides a Web-based forum where you can join other InstallShield users, post questions, and search for answers. • **Webinars**—Directs you to free Web-based seminars that help you evaluate InstallShield and gain the most from your Flexera Software products. • **Downloads**—Offers a place where you can download the latest InstallShield prerequisites, InstallShield merge modules, and objects; service packs; patches; and more for the version of InstallShield that you are using. • **Release Notes**—Connects you to the release notes that are posted to the Knowledge Base on the Flexera Software Web site. • **Known Issues**—Displays the Knowledge Base article that lists the known issues for the version of InstallShield that you are using and provides information about workarounds and resolutions. • **Upgrade Alerts**—Displays the Knowledge Base article that provides information about possible issues that may occur when you upgrade projects that were created with earlier versions of InstallShield to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from earlier versions. • **RSS Feeds**—Directs you to the Web page where you can subscribe to RSS feeds of InstallShield Knowledge Base articles.
Contact Us	To access the Support area of the Flexera Software Web site or join the Customer Experience Improvement Program, click one of the links listed in this section.

Working with Projects

An InstallShield project specifies the files, folders, and operations that make up the project's output. A project's output is one of the following (depending on the project type):

- Installation: creates or updates a copy of your application on the target system
- Redistributable (merge module or InstallShield object): contains logic and files needed to install distinct pieces of functionality when included in an installation
- Transform: enables an administrator to apply modified settings to an Windows Installer installation when deploying the installation

A project can be as simple or as complex as you need to meet your requirements. A simple project might consist of files, components, features, and registry entries. More complex projects might consist of these items plus redistributables, initialization file changes, and calls to external DLL functions.

Installation Projects Overview

With InstallShield, you have the ability to choose between a variety of different installation project types—those that use InstallShield's powerful InstallScript programming language (InstallScript), those that use the Windows Installer database (Basic MSI), or a combination of the two (InstallScript MSI).

The installation project type you choose depends on the needs of your software installation. InstallShield provides the same intuitive user interface for all project types, so your application's needs can dictate the project type and your authoring experience does not change. Additionally, the knowledge gained by learning how to create projects of one type is applicable to projects of the other types.

Similarities Between Project Types

All InstallShield installation project types create installations that have the same professional, industry-standard look and feel that your end users have come to expect from InstallShield. All enable you to create customized installations to meet the needs and expectations of your end users.

Despite the similarities in the project types' feature sets, there are some important differences between them—especially between the InstallScript and Basic MSI projects. See Determining Which Installation Project Is Right for You to learn which of these project types is right for your software installation needs.

Determining Which Installation Project Is Right for You

InstallShield enables you to create different types of projects, but for the beginner there is just one significant choice to make: Do you want to build your installation with Windows Installer or InstallScript technology? Deciding which project type is right for you depends on your experience with InstallShield software and installation development and on your installation and deployment needs. The three main project types are described below.

Basic MSI Projects

Basic MSI projects use the Windows Installer service to drive the entire installation, including calls to any custom actions (InstallScript, VBScript, JScript, .exe files, .dll files, managed code). The Basic MSI project type is recommended when you want to do any of the following:

- You want to meet the Windows logo program requirements.
- You want to maximize compatibility with administrative tools such as Microsoft System Center Configuration Manager, or if your software will be customized by corporate system administrators prior to deployment. The Basic MSI project type gives them the flexibility to create transforms for the installation package and its associated properties, without repackaging your installation.
- You want to avoid writing scripting code and want to instead set properties and make table entries.
- You want to upgrade an existing Basic MSI project.

InstallScript Projects

InstallScript projects use InstallScript to control the installation. This project type is recommended for any of the following scenarios:

- You have advanced requirements for the end-user experience (the end-user dialogs), such as multimedia elements.

- You want to use full-screen billboards during the run time of your installation.
- You prefer authoring the project using a procedural language at its core rather than a set of database tables.
- You want to perform actions before or after the main installation has been run.
- You want to upgrade an existing InstallScript project.

InstallScript MSI Projects

InstallScript MSI projects use both the Windows Installer engine and the InstallScript engine to drive the installation. This project type is the most complicated since it uses both of those engines for the installation; it is not recommended for beginners. This project type may be recommended in the following scenarios:

- You want to meet the Windows logo program requirements.
- You have advanced requirements for the end-user experience (the end-user dialogs), such as multimedia elements.
- You prefer authoring the project using a procedural language at its core rather than a set of database tables.
- You want to perform actions before or after the main installation has been run.
- You want to upgrade an existing InstallScript MSI project.



Tip *• Repackager is a project conversion tool that is available in AdminStudio. You can use this tool to convert InstallScript projects and InstallScript MSI projects to Basic MSI projects.*

Project Types

InstallShield offers a number of different project types to assist you in creating the optimal project for your end users.

Table 2-3 • Project Type Descriptions

Project Type	Icon	Description
Advanced UI		 ***Edition*** *• This project type is available in the Professional edition of InstallShield. If you open this type of project in the Premier edition of InstallShield, it opens as a Suite/Advanced UI project.* An Advanced UI project lets you create a bootstrap application with a contemporary, customizable user interface for a single .msi package, .msp package, or InstallScript package, as well as multiple InstallShield prerequisites. An Advanced UI installation uses a setup launcher (**Setup.exe**) to conditionally launch packages on target systems as needed.

Table 2-3 • Project Type Descriptions (cont.)

Project Type	Icon	Description
Basic MSI		A Basic MSI project uses Windows Installer to provide the user interface for the installation. When you choose this project type, you need to create features and components, and specify all application files and other distributable data.
DIM		A DIM project is targeted toward engineering teams who want to foster collaboration, enable distributed installation development, and maximize efficiency in their organizations. A developer installation manifest (DIM) is a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete, logically separated portion of a product installation. You can incorporate one or more DIMs into Basic MSI projects.
InstallScript		The InstallScript installation project provides the flexibility afforded by the InstallScript language and is driven by the proven InstallScript engine.
InstallScript MSI		An InstallScript MSI project uses both InstallScript and Windows Installer tables. With this project type, you need to create features and components, and specify all application files and other distributable data.
InstallScript Object		Using the InstallScript Object project, you can create an InstallShield merge module that uses InstallScript. This type of merge module, however, can be consumed only by InstallShield.
Merge Module		Select this option to create your own merge modules. You need to create any components and file associations.
MSI Database		Select MSI Database to edit your installation project in Direct Edit mode. This means that you can directly edit the Windows Installer tables within InstallShield to configure your installation.
MSM Database		Select MSM Database to edit your merge module in Direct Edit mode. This means that you can directly edit the Windows Installer tables within InstallShield to configure your merge module.
QuickPatch		This project type is recommended for installation authors who want to ship small, single updates to their end users. QuickPatch authoring provides an alternative to creating a patch configuration in the Patch Design view even though it provides less customization. Creation of a QuickPatch begins with the Create New QuickPatch Wizard.

Table 2-3 • Project Type Descriptions (cont.)

Project Type	Icon	Description
Suite/Advanced UI		***Edition*** • *This project type is available in the Premier edition of InstallShield.* A Suite/Advanced UI project lets you create a bootstrap application with a contemporary, customizable user interface for multiple .msi packages, .msp packages, InstallScript packages, .exe packages, Web Deploy packages, sideloading UWP app packages (.appx), and Windows Installer transactions, as well as multiple InstallShield prerequisites. A Suite/Advanced UI installation packages together multiple separate installations as a single installation while providing a unified user interface; it uses a setup launcher (**Setup.exe**) to conditionally launch packages on target systems as needed.
Transform		A transform (.mst file) is a simplified Windows Installer database that contains the differences between two Windows Installer databases. Creating a new transform project launches the Open Transform Wizard.
Visual Basic .NET Wizard		Select this option to launch the Visual Studio .NET Wizard for Visual Basic .NET, C# .NET, and Visual C++ .NET.
C# .NET Wizard		Select this option to launch the Visual Studio .NET Wizard for Visual Basic .NET, C# .NET, and Visual C++ .NET.
Visual C++ .NET Wizard		Select this option to launch the Visual Studio .NET Wizard for Visual Basic .NET, C# .NET, and Visual C++ .NET.

Advanced UI and Suite/Advanced UI Projects

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Advanced UI and Suite/Advanced UI installations are bootstrap applications that package together installations and InstallShield prerequisites as a single installation while providing a unified, fully customizable user interface. They use a setup launcher (**Setup.exe**) to conditionally launch packages on target systems as needed.

Note that Advanced UI and Suite/Advanced UI installations require Windows Vista or later or Windows Server 2008 or later.

Also note that the ability to create and build a Suite/Advanced UI installation that includes a sideloading UWP app package (.appx) requires Windows Vista or later or Windows Server 2008 or later on the machine that has InstallShield or the Standalone Build.

Basic MSI Installation Projects

Basic MSI is the Windows Installer–based project type in InstallShield. Basic MSI projects are recommended in cases where the Windows Installer service should drive the entire installation. They allow you to author your installation using only the native Windows Installer feature set. The geometry of your end user dialogs as well as the flow of your setup user interface (UI) is authored directly in the MSI package, and the Windows Installer Service uses its native user interface rendering capabilities to display the UI to your end users. The advantages of this project type are fully realized if you need to author your installation in an open format.



***Note** • Create a Basic MSI project if your software will be customized by corporate systems administrators prior to deployment. It gives them the flexibility to create transforms for the installation package and its associated properties, without repackaging your installation.*

Basic MSI projects have the ability to run InstallScript code in the form of custom actions. As with InstallScript projects, Basic MSI projects take advantage of other robust features provided by InstallShield such as dialog authoring, the ability to call custom actions in standard Windows .dll files, and the ability to specify support files.

Run-Time Behavior for Basic MSI Installations with InstallScript Custom Actions

For InstallShield 12 and Later Projects

The following steps outline the run-time behavior for Basic MSI installations that include InstallScript custom actions:

1. The package is launched by the Windows Installer service.
2. Any sequenced InstallScript custom actions execute as follows.
    a. The Windows Installer calls the relevant entry point in **ISSetup.dll** (loaded from the **Binary** table).
    b. **ISSetup.dll** extracts its **Setup.inx** resource to a temporary location.
    c. **ISSetup.dll** executes relevant script code.
    d. Windows Installer unloads **ISSetup.dll**.

The run-time behavior for InstallShield 12 and later Basic MSI projects that have InstallScript custom actions is much different than the behavior for InstallShield 11.5 and earlier because of some architecture enhancements. For more information about the enhancements, see Upgrading Projects from InstallShield 11.5 or Earlier.

For InstallShield 11.5 and Earlier Projects

For installations created with InstallShield 11.5 and earlier, the run-time behavior is as follows:

1. **Setup.exe** installs **ISScript.msi**.
2. **Setup.exe** launches **MsiExec.exe** to install the primary .msi file. (**Setup.exe** must be included in the installation, or it must already be present on the target system.)
3. ISMsiServerStartup (immediate custom action) starts and initializes **IDriver.exe**.
4. Any sequenced InstallScript custom actions execute as follows.
    a. The Windows Installer finds the relevant entry point in **ISScriptBridge.dll**.
    b. **ISScriptBridge.dll** finds the running **IDriver.exe** instance using the running object table (ROT).

c. IDriver.exe executes relevant script code.

5. ISCleanUpSuccess shuts down **IDriver.exe**.

If you upgrade an InstallShield 11.5 or earlier project to InstallShield 2016, the run-time behavior is updated to the behavior described for InstallShield 12 and later projects.

DIM Projects

The DIM support in InstallShield is targeted toward engineering teams who want to foster collaboration, enable distributed installation development, and maximize efficiency in their organizations. A developer installation manifest (DIM) is a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete, logically separated portion of a product installation. Some benefits of using DIMs are as follows:

- Working with DIMs enables multiple team members to contribute to the development of the installation simultaneously. Each software developer or other team member can work on a separate DIM that the release engineer can reference in one or more installation projects.
- Release engineers can reuse DIMs in multiple installation projects, enabling efficiency.
- DIMs include support for virtually the same functionality that is available in Basic MSI projects. This gives authors of DIMs all of the flexibility that they need to develop their portions of an installation.

InstallScript Installation Projects

The InstallScript installation project type provides the power and flexibility of the InstallScript programming language and the familiarity of the InstallShield Professional project. Windows Installer is not used at all. The installation is completely script driven and very flexible, and it uses the InstallScript engine, the same trusted engine used by InstallShield Professional.



Note • *InstallScript installations are generally less desirable to systems administrators because they require the use of Setup.exe, rather than being self-contained in an .msi file, and may not be fully customizable prior to deployment. If your software will be customized by corporate systems administrators prior to deployment, create a Basic MSI installation project.*

The InstallScript Engine

With InstallScript projects, the run-time user interface is rendered and its flow controlled by InstallScript, and the changes that are made to the target operating system are done through InstallShield's trusted InstallScript engine, with no reliance on Windows Installer.

Using InstallScript as the installation driver has many benefits. The first is that InstallScript's event model enables you to create a script-driven installation without writing a single line of code. If you want to add custom functionality, you need to implement only the events whose functionality you want to change.

The user interface abilities for Basic MSI projects are somewhat limited with regard to the types of controls you can use or the kind of control you have over dialog events. InstallScript projects have no such limitation—offering a wide range of standard controls along with the ability to add custom dialog controls. Dialog messaging in the script enables you to have complete control over how your end-user dialogs behave.

Upgrading Projects Created Using InstallShield Professional

Installation projects created using InstallShield Professional (version 5.5 and later) upgrade to the InstallScript project type.

Additional Features for InstallScript Projects

InstallScript projects provide a few additional features that are not available in Basic MSI projects:

Table 2-4 • Additional InstallScript Project Features

Feature	Description
Setup Types view	This view enables you to easily create different installation configurations for your application. This view also lets you select the defaults for the Custom setup type.
Billboard Support	Run-time billboard support is available only for InstallScript projects. Billboards let you show bitmaps to the end user while files are being transferred to their machine. You can use these bitmaps to entertain or educate user.

InstallScript MSI Installation Projects

InstallScript MSI installations use two different installer engines:

- The Windows Installer engine runs the standard Execute sequence of the .msi package. This is the sequence that typically modifies the target system.
- The InstallScript engine serves as the custom user interface (UI) handler of the installation. It also executes the InstallScript code. The advantage of using the InstallScript engine for the UI is that it offers support for highly customized run-time dialogs.

In addition, InstallShield offers two different styles for the UI of InstallScript MSI installations:

- **Traditional style**—This style lets you use the InstallScript engine as an external UI handler for your InstallScript MSI installation. With this style, your installation must include a **Setup.exe** setup launcher. The setup launcher serves as a bootstrap application that initiates the InstallScript engine to display the UI and run the InstallScript code, and the Windows Installer to run the Execute sequence of the .msi package.
- **New style**—This style lets you use the InstallScript engine as an embedded UI handler for your InstallScript MSI installation. For this style, InstallShield embeds the InstallScript engine within the .msi package. The Windows Installer calls the InstallScript engine to display the UI. The Windows Installer also runs the Execute sequence of the .msi package.

  This option requires Windows Installer 4.5 on the target machine. This option also has some limitations that require careful planning if you decide to use this style.

For detailed information about these two styles, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

Note • *Because this project type uses two different engines, it is more complex than pure InstallScript or Basic MSI installation projects. It is recommended only for advanced users.*

Traditional-style InstallScript MSI installations are less desirable to systems administrators than Basic MSI installations or new-style InstallScript MSI installations. That is because traditional-style InstallScript MSI installations require the use of **Setup.exe***, rather than being self-contained in an .msi package, and they may not be fully customizable prior to deployment. If your software will be customized by corporate systems administrators prior to deployment, create a Basic MSI project. As an alternative, you could consider using the new-style InstallScript MSI installations, but your must ensure that any system changes that need to be made are done with custom actions in the Installation Execute sequence. To ensure that sufficient privileges are available, custom actions should have an in-script execution setting of deferred in system context.*

Additional Features for InstallScript MSI Projects

InstallScript MSI projects provide a few additional features that are not available in Basic MSI projects:

Table 2-5 • Additional InstallScript MSI Project Features

Feature	Description
Setup Types view	This view enables you to easily create different predefined installation configurations for your application, such as Typical or Compact. This view also enables you to select the defaults for the Custom setup type.
Billboard Support	Run-time billboard support is available for InstallScript MSI installations, but not Basic MSI installations. Billboards enable you to show bitmaps while files are being transferred to an end user's machine. You can use these bitmaps to entertain or educate end users.

Run-Time Behavior for InstallScript MSI Installations

For InstallShield 12 and Later Projects

The following steps outline the run-time behavior for InstallScript MSI installations:

1. **Setup.exe** initializes **ISSetup.dll**.
2. **ISSetup.dll** does the following:
    a. Loads **Setup.inx**.
    b. Executes pre-installation InstallScript events (for example, OnBegin).
    c. Launches the Windows Installer (and the InstallScript engine for the external user interface).
    d. Each InstallShield custom action executes as follows (for example, OnMoved). (Note that this essentially the same mechanism that is used in Basic MSI with InstallScript custom actions.)
        i. The Windows Installer calls the relevant entry point in **ISSetup.dll**. (This is a different instance of **ISSetup.dll**, loaded from the **Binary** table.)
        ii. **ISSetup.dll** extracts its **Setup.inx** resource to a temporary location.

iii. **ISSetup.dll** executes relevant script code.

iv. Executes post-installation InstallScript events (for example, OnEnd).

The run-time behavior for InstallShield 12 and later InstallScript MSI projects is much different than the behavior for InstallShield 11.5 and earlier because of some architecture enhancements. For more information about the enhancements, see Upgrading Projects from InstallShield 11.5 or Earlier.

For InstallShield 11.5 and Earlier Projects

For InstallScript MSI installations created with InstallShield 11.5 and earlier, the run-time behavior is as follows:

1. **Setup.exe** installs **ISScript.msi**.
2. **Setup.exe** launches **IDriver.exe**.
3. **IDriver.exe** does the following:
   a. Loads **Setup.inx**.
   b. Executes pre-installation InstallScript events (for example, OnBegin).
   c. Launches the Windows Installer (and the InstallScript engine for the external user interface).

      Each InstallScript custom action executes as follows (for example, OnMoved). (Note that this is the same mechanism used in Basic MSI installations with custom actions.)

      i. The Windows Installer calls the relevant entry point in **ISScriptBridge.dll**.

      ii. **ISScriptBridge.dll** finds the running **IDriver.exe** instance using the running object table (ROT).

      iii. **IDriver.exe** executes relevant script code.

   d. Executes post-installation InstallScript events (for example, OnEnd).

Merge Module Projects

Merge modules allow you to add existing pieces of functionality to your installation. For example, if your application requires the VB Run-Time DLL, you can add Microsoft's Visual Basic Virtual Machine module to your setup application. Traditionally, you would have to create registry entries, set the target folders, custom actions and any other necessary steps yourself. With merge modules, all you need to do is associate an existing merge module with your installation and move on to your next task.

InstallShield provides you with the ability to create your own merge modules to distribute with your installations or give away to other developers. This can be useful if you are going to include certain pieces of functionality in all of your installation projects. Rather than having to recreate that part of the installation every time, you can create it once as a merge module and then associate that module with all of your installation projects.

InstallScript Custom Actions in Merge Module Projects

Merge modules can contain InstallScript custom actions. However, you must manually create the **Module**Sequence** table entries to make sure that the custom actions are merged properly.

How Merge Modules Work

When a merge module is associated with an installation project, it is merged into the .msi database at build time. As a result, you are merging two distinct projects (your new installation and the existing merge module) into one .msi file. The merge module behaves like a component in that it is not installed unless the feature with which it is associated is installed. Therefore, if you have a feature that requires VB run-time DLLs, and that feature is not selected for installation, the VB merge module will not be installed.

MSI Database and MSM Database Projects

The MSI Database and MSM Database project types allow you to edit Windows Installer installation databases and merge module databases directly, rather than working through an intermediate project format (.ism file). These project types extend the Direct Editor functionality to include different views that are supported in standard installation and merge module projects. The views that are supported in these project types are intended to function as they do for an .ism project.



Project • *There is no string entry or path variable support in an MSI Database or MSM Database project.*

You can directly edit an existing .msi file or .msm file, or directly create a new database by opening a new MSI Database or MSM Database project. The data in this file should match what you would get if you created a blank Basic MSI or Merge Module project and built a database from that project.

InstallScript Object Projects

Using InstallShield objects, you can easily install key Windows technologies. To install a technology, simply include the corresponding InstallShield object in your installation. If a technology is required by a particular component, you can associate the corresponding InstallShield object with the feature.

QuickPatch Projects

A QuickPatch project is a specific type of project recommended for installation authors who want to ship small, single updates to their users. Changes that are more extensive such as adding custom actions and changing .ini data typically require a standard patch.

QuickPatch authoring provides a simple alternative to creating a patch configuration in the Patch Design view, even though it provides less customization. Fundamentally, both patch creation methods produce the same deliverable types: .msp and .exe files.

With a QuickPatch, you can do any of the following:

- Add new files to the original installation or an earlier QuickPatch.
- Delete files in the original installation.
- Delete files that were added with a previous QuickPatch.
- Perform the same set of above operations on registry entries.
- Remove custom actions that were included with the original installation but that do not apply to the current QuickPatch project.

The creation of a QuickPatch project always begins with the Create New QuickPatch Wizard. After you have completed the wizard, you can configure the QuickPatch project settings once the project opens in InstallShield.

Transform Projects

A transform (.mst file) is a simplified Windows Installer database that contains the differences between two .msi databases. Transforms enable an administrator to apply modified settings to a database when deploying an installation package.

InstallShield has a transform project type that enables you to edit .msi packages without having to convert them to an InstallShield (.ism) project. You can save the changes made as a transform (.mst) file.

When you create a transform project or open a transform in InstallShield, the Open Transform Wizard launches to gather information about the base .msi file and any additional transform files that you want applied to the base .msi file.

Using Projects

InstallShield allows you to create, edit, upgrade, and save numerous project types—from installation projects to InstallShield object projects that allow you to reuse functionality within InstallShield.

The pages in this section cover a variety of topics, including how to create a particular project type, how to create project templates, and what to expect when you upgrade from a different InstallShield product.

Creating New Projects

There are a number of ways to create a new InstallShield project.



Task ***To create a new project, do one of the following:***

- Click the **New Project** button on the toolbar or on the InstallShield Start Page.
- Press Ctrl+N.
- On the **File** menu, click **New**.

Any of these steps launches the New Project dialog box, from which you can select the project that you want to create.

Using a Template to Create a New Project

Your project templates are listed in the New Project dialog box on the appropriate tab. For more information, see Basing New Projects on Templates.

Creating a Project From Within Microsoft Visual Studio

You can create an InstallShield project from within the Microsoft Visual Studio workspace. For more information, see Creating InstallShield Projects in Microsoft Visual Studio.

Opening Projects

Task ***To open an existing InstallShield project, do one of the following:***

- Click the **Open Project** button on the toolbar.
- On the **File** menu, click **Open**. In the **Open** dialog box, navigate to the project file.
- Press Ctrl+O.
- Click the **Open an existing project** link or click a recently opened project link on the InstallShield Start Page.
- Double-click a project file (.ism) on the desktop or in Windows Explorer.

With the exception of clicking a specific file or file link, all of the above options launch the Open dialog box, which enables you to browse to your project file.

Opening Projects from Source Code Control

An option on the Open dialog box enables you to get the latest version of a project folder from your source control application.

Task ***To open a project from source control:***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. Click the **Source Control** button. A login dialog box for your source control program opens.
3. Log on to your source control program and browse to your project folder or database.

InstallShield gets the latest versions of every file in that project to a working directory.

Note • *If you do not have a source control application on the system that has InstallShield, the Open dialog box does not display the Source Control button.*

Opening Patch Creation Properties Files in Direct Edit Mode

You can open a previously created patch creation properties file from the Open dialog box.

Task ***To open a patch creation properties (.pcp file):***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. In the **Files of type** list, click **Patch Creation Properties Files (*.pcp)**.
3. Select the .pcp file you want to edit and click **Open**.

Your patch project file opens in the Direct Editor.

Opening Windows Installer Packages

InstallShield provides the option to either edit MSI packages directly in the IDE in Direct Edit Mode or convert it to an InstallShield project (.ism), and then edit it as a Basic MSI project in the IDE.



Note • *Some functionality may be limited when editing an MSI package in Direct Edit Mode.*



Task ***To open an MSI package in Direct Edit Mode of the IDE:***

1. Click the **Open Project** button on the toolbar. A browse dialog opens.
2. Select **Windows Installer Packages (*.msi)** from the **Files of Type** list.
3. Select the MSI package that you want to open.
4. Ensure that the **Open as** field is set to Auto, and click **Open**.



Note • *By default, the IDE opens an MSI package in Direct Edit Mode.*

The Open MSI/MSM Wizard can convert an existing Windows installer (.msi) package to an InstallShield project (.ism) file, and open the new project to modify in the IDE.



Note • *You can also open Patch Creation Project files (.pcp files) in InstallShield and edit them in the Direct Editor.*

Opening Merge Modules

The Open MSI/MSM Wizard can convert an existing Windows Installer merge module (.msm file) to an InstallShield project (.ism) file, and open the new project in the InstallShield IDE.



Task ***To launch the Open MSI/MSM Wizard:***

1. Click the **Open Project** button on the toolbar. A browse dialog opens.
2. Select **Windows Installer Modules (*.msm)** from the **Files of Type** list.
3. Go to the merge module and click **Open**.

The Open MSI/MSM Wizard prompts for specific information about the project you are creating, and then opens your new merge module project in the IDE.

Opening Object Projects

You can open an object project (.ipo or .ipr) that you created using InstallShield Professional. When you open this project in InstallShield, it is migrated to a merge module (.msm) project.

Placing Merge Modules in the Redistributables View

To associate your merge module with InstallShield installation projects, you can automatically place your merge module project in the Modules folder so it is available in the Redistributables view for use in setup projects.



Task ***To automatically place your merge module project in the Modules folder:***

Select the "Add to local merge module catalog" option in the Merge Module Options panel of the Release Wizard.

Saving Projects

When you create a new project, it is saved automatically with the name and location that you provide in the New Project dialog box.

InstallShield enables you to save a copy of the open project as a template or as a new project with a different name and location.

Saving a Project with a New Name and Location

When you save your project with a new name and location, a copy of the renamed project file and all of its associated files and folders are saved in the new location. The next time that you save your project, all changes are saved to this new location.



Task ***To save your project with a new name and location:***

1. On the **File** menu, click **Save As**. The **Save As** dialog box opens.
2. In the **Save in** box, select the appropriate location.
3. Select or clear the **Create 'Project Name' subfolder and save the project in the created folder** check box as appropriate. In both cases, a <*Project Name*> subfolder is created in the **Project Location** box (on the **File Locations** tab of the **Options** dialog box).

   If you select the **Create 'Project Name' subfolder and save the project in the created folder** check box, the project file (.ism) is saved in the <*Project Name*> subfolder. If you clear the check box, the project file is saved in the location specified in the **Project Location** box.
4. If the new project's GUID should be different than the GUID of the original project, select the **Create and assign a new project GUID to the saved projec**t check box. If the new project's GUID should be the same GUID of the original project, clear this check box.
5. To use the name that you specified in the **File name** box as the value for the **Product Name** setting (in the **General Information** view) of the new project, select the **Update the project settings appropriately based on the new project name** check box. If the new project's **Name** property should be the same as that of the original project, clear this check box.



Note • *When you save a project with external dependencies, such as InstallScript files, your new project points to the original copies of these files. Duplicate copies are not made. Therefore, before you delete your original project, ensure that you do not delete any files that may be used by the new project.*

Converting from One Project Type to Another Project Type

InstallShield offers support for converting some types of projects to other types of projects.

Table 2-6 • Project Converters

Project Converter	How-To Information
Convert Basic MSI project to InstallScript MSI project	See Converting a Basic MSI Project to an InstallScript MSI Project.
Convert InstallScript MSI project to InstallScript project	See Converting an InstallScript MSI Project to an InstallScript Project.
Convert InstallScript project to Basic MSI project	Use Repackager to convert an InstallScript project to a Basic MSI project. ***Edition*** • *Repackager is included with AdminStudio.*
Convert InstallScript MSI project to Basic MSI project	Use Repackager to convert an InstallScript MSI project to a Basic MSI project. ***Edition*** • *Repackager is included with AdminStudio.*
Convert InstallScript project to InstallScript MSI project	This conversion is a two-step process: **1.** Use Repackager to convert an InstallScript project to a Basic MSI project. **2.** Use InstallShield to convert the Basic MSI project to an InstallScript MSI project. See Converting a Basic MSI Project to an InstallScript MSI Project. ***Edition*** • *Repackager is included with AdminStudio.*
Convert Visual Studio setup project to InstallShield Basic MSI project	See Converting or Importing Visual Studio Projects into InstallShield Projects.
Convert Visual Studio merge module project to InstallShield Merge Module project	See Converting or Importing Visual Studio Projects into InstallShield Projects.

Converting a Basic MSI Project to an InstallScript MSI Project

Converting your Basic MSI project to an InstallScript MSI project allows you to take full advantage of the scripting capabilities that are available in InstallShield.



Caution • *This conversion is irreversible. After you convert your Basic MSI project to an InstallScript MSI project and save it, you cannot easily convert your project back to a Basic MSI project. Before you convert your Basic MSI project, it is generally good practice to first create a back-up copy of it.*

Converting Projects



Task ***To convert your Basic MSI project to an InstallScript MSI project:***

1. Open your Basic MSI project.
2. On the **Project** menu, point to **Project Converters**, and click **Convert to InstallScript MSI project**.

Converting Dialogs

Because an InstallScript MSI project uses resource-based dialogs—instead of using the Windows Installer dialog–related tables—no Windows Installer–based dialogs from your original project are available after the migration. You need to manually add any custom dialogs.

Your converted installation project contains the default InstallScript MSI dialogs. To review the default dialogs, go to the InstallScript view and view the dialogs displayed in the OnFirstUIBefore and OnFirstUIAfter events.

Converting an InstallScript MSI Project to an InstallScript Project

InstallScript MSI projects created using InstallShield, InstallShield DevStudio, or InstallShield Developer can be easily converted to the InstallScript project type.



Task ***To convert an existing InstallScript MSI project to an InstallScript project:***

1. Open the InstallScript MSI project.
2. On the **Project** menu, point to **Project Converters**, and click **Convert to InstallScript project**.

GUIDs

GUID stands for *globally unique identifier*. A GUID is 128 bits long, and the algorithm used to generate a GUID guarantees each GUID to be unique. Because GUIDs are guaranteed to be unique, they can be used to identify COM classes, Product Codes, and various other codes.

For example, after a product is installed, a key is created under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall and is named after the installation's Product Code. At one time, this key was named after the Product Name. However, this caused a potential conflict. If two installations were installed on the same machine, and both shared the same Product Name, they would also share the same registry key. Because a GUID is now used, it guarantees that this conflict does not occur.

An example GUID is {5D607F6A-AF48-4003-AFA8-69E019A4496F}. Any letters in a GUID must be in uppercase.

GUIDs in Your Project

When you create an installation project, there are a number of different GUIDs that are relevant to your project.

Table 2-7 • GUIDs

GUID Name	Description
Product Code	The product GUID uniquely identifies your application.
Package Code	The package code uniquely identifies your installation package. ***Project*** *• This GUID is available in the following project types: Basic MSI, InstallScript MSI, and QuickPatch.*
Package GUID	The package GUID uniquely identifies the installation package in the Advanced UI or Suite/Advanced UI installation. ***Project*** *• This GUID is available in Advanced UI and Suite/Advanced UI projects.*
Patch GUID	The patch GUID uniquely identifies a patch package. ***Project*** *• This GUID is available in the following project types: Basic MSI, InstallScript MSI, and QuickPatch.*
Suite GUID	The Suite GUID uniquely identifies your Advanced UI or Suite/Advanced UI installation. ***Project*** *• This GUID is available in Advanced UI and Suite/Advanced UI projects.*
Upgrade Code or Upgrade GUID	The upgrade GUID uniquely identifies a family of products for upgrade purposes. It is important for major upgrades. ***Project*** *• This GUID is available in the following project types: Basic MSI, InstallScript MSI, and QuickPatch.*

Project *• For information on when you need to change GUIDs in a Basic MSI or InstallScript MSI project, see Major Upgrade vs. Minor Upgrade vs. Small Update.*

Changing the Default Project Location

All new projects are saved by default in the following location:

C:\InstallShield 2016 Projects

To specify a new default location for your installation projects, use the **File Locations** tab on the **Options** dialog box.



***Note** • Although this folder is used for all new installation projects, any existing projects remain in the previous location.*

Reusing Project Elements

You can save several elements in your project to reuse in another project. In some cases, the data must be saved to an intermediary file and in others you can export it directly to another .ism file. The manner in which you can reuse each element is described below.

Entire Projects

By saving a project as an InstallShield project template, you can create a project from the template that is almost identical to the original project.

Registry Entries

When you export registry entries to a REG file, you can import that file into another component's registry data in the same project or an entirely different one.

String Entries

You can export a project's string entries for a specific language to a tab-delimited text file. Typically, you would send the text file out for translation and then import the string entries back into the project for another language. However, you could also add the exported string entries to another project by importing the file into the other project.

End-User Dialogs

You can save all of your dialogs to an .rc file, export them individually to InstallShield dialog template files, or export them directly to another project.

Components

To export a component to another project, right-click on the component and select Export Components Wizard. Follow through wizard and the component and all of its data are added to the specified project.

Custom Actions

You can export a custom action to another project by right-clicking on the custom action in the Custom Actions explorer and clicking Export. The Custom Actions explorer is in the Custom Actions and Sequences view (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) and the Custom Actions view (in DIM, Merge Module, and MSM Database projects).

Using a Repository to Share Project Elements

InstallShield supports the use of repositories. A repository is a collection of common elements that can be shared and reused in different installation projects. Elements that can be stored in a repository include:

- End-user dialogs
- InstallScript files
- Merge modules
- Project templates
- SQL scripts
- System searches

Repositories provide you with the ability to reuse project elements in multiple projects and strive for consistency. They also save you from having to duplicate work. For example, if many of the installations for your organization's products include a particular custom dialog, you can create that custom dialog once and then publish it to your repository. Any time you want to use that dialog in another installation, simply add the dialog from your repository to your project.

Two types of repositories are available in InstallShield:

- Local repository—A local repository is your own collection of installation elements that you want to be able to reuse in multiple projects. A local repository is stored on your local machine, and it is *not* available to other installation authors.
- Network repository—A network repository is a collection of installation elements that multiple installation authors can access and reuse in their projects as needed. A network repository fosters collaboration among installation authors; it is stored on a network. For information on configuring a network repository, see Setting up a Network Repository.



Edition • *Network repositories are available in the Premier edition of InstallShield only. Local repositories are available in both the Premier and Professional editions of InstallShield.*

Setting up a Network Repository



Edition • *Network repositories are available in the Premier edition of InstallShield only.*

To configure a network repository, or to change the network repository settings, use the Repository tab on the Options dialog box.

Project Templates

A project template contains all of the default settings and design elements that you want to use as a starting point when you create an installation project or merge module project.

You can open a template in InstallShield and edit it as you would a project. To create a template, save any project as a template. For more information, see Creating Project Templates.

For instructions on starting a new project based on a template, see Basing New Projects on Templates.

InstallShield project templates serve only as a starting point for new projects. After you have created a project based on a template, there is no link between the current project and the existing template. If you change an element in the template, it does not affect the project that you created based on that template. However, you can modify the template and create another project based on the updated version of the template.

Creating Project Templates

The procedure for creating a template file varies, depending on whether you are creating a template for a project type that uses .ism project files (Basic MSI, InstallScript, InstallScript MSI, Merge Module), or for a project type that uses .issuite project files.

Creating a Project Template File for Projects that Use .ism Project Files

You can create a project template from any .ism project file; this includes installation projects and merge module projects. When you create a template from an .ism project file, the template file has an .ist file extension.



Task ***To create a template (.ist) from an .ism project file:***

1. Edit the project to include the desired default settings, user interface, features, and so on.
2. On the **File** menu, click **Save As**. The **Save As** dialog box opens.
3. In the **Save As** type box, select **InstallShield Template (*.ist)**.
4. Type a file name (with the .ist extension) and select a location for the new template. The recommended location for templates is a Templates folder within the project location folder.
5. Click **OK**.

You can open the template in InstallShield and make additional changes. You can use the final version of your template to create a new project.

Creating a Project Template File for Projects that Use .issuite Project Files

You can designate any Advanced UI or Suite Advanced UI project (.issuite) to be a template. Project files and template files for these project types have the same file extension: .issuite.



Task ***To designate that an .issuite project file is a template file and make the template available for selection in the New Project dialog box:***

Right-click in the **All Types** tab on the **New Project** dialog box and then click **Add New Template**. InstallShield lets you browse to the .issuite file that you want to use as a template, and it adds a new icon for it to the **All Types** tab.

Basing New Projects on Templates

To save time, you can base multiple projects on a single project template that contains default settings and design elements. The template can be stored somewhere on your system. It can also be stored in a repository.



Task ***To create a project using a template as a starting point:***

1. On the **File** menu, click **New**. The **New Project** dialog box opens.
2. Click the **All Types** tab.
3. Select the template that you want to use.

   Note *• To display templates that are stored in a repository, right-click in the box of project types and ensure that* ***Show Templates in Repository*** *is enabled. To add a template to the tab, right-click in the box of project types and then click* ***Add New Template****. Then browse to the desired template file (.ist file).*
4. In the **Project Name** box, type a name for your project.
5. In the **Location** box, type the path to the directory where you want to store your template, or click the **Browse** button to find it.
6. Click **OK**. InstallShield creates the new project based on the template that you selected, and the new project is opened in InstallShield.

The new project is virtually identical to the project template, except that InstallShield generates a new product code (for installation projects), package code, and upgrade code (for Windows Installer–based installation projects), and a module ID for merge module projects.

Publishing Project Templates to a Repository

If you have an existing template that you would like to reuse for other projects or share with other users, you can publish it to a repository.



Task ***To publish a template to a repository:***

1. On the **File** menu, click **New**. The **New Project** dialog box opens.
2. Click the **All Types** tab.
3. Right-click the template that you want to publish and then click **Publish Template to Repository**. The **Publish Wizard** opens.
4. Complete the panels in the Publish Wizard.

After you have created a project that is based on a template in your repository, there is no link between the current project and the existing repository template. If you make a change to the template and then republish it to the repository, it does not affect the project that you created based on the template. However, you can create a new project based on the updated version of the template in the repository.

Sample Projects

A number of example project (.ism) files have been included with the InstallShield installation. These projects are stored in the Samples folder. The default location is:

C:\Program Files\InstallShield\2016\Samples

These projects show you how a certain aspect of an installation project—such as release flags—can be implemented.

Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

InstallShield provides a Project Assistant to help you quickly and easily build a basic installation project. The Project Assistant provides a framework of installation project tasks to guide you through the project creation process and provides pertinent information along the way.

How the Project Assistant Works

When you create a new installation project or transform, the Project Assistant view automatically opens.

Information that you enter in the Project Assistant is saved directly to the underlying project file. Therefore, you can switch to the Installation Designer (described below) and view or modify your information using the full power of the InstallShield Installation Development Environment (IDE). In this way, you can use the Project Assistant to create the foundation for a more advanced installation where you use the Installation Designer, as needed, to customize your installation.

Integration with the Installation Designer

The Installation Designer tab displays all of the views in the InstallShield interface that are available for your project type. Here you can add more complex and powerful elements to your installation project. You can create your installation project using the Project Assistant and then use the Installation Designer to fine-tune project elements.

The Installation Designer and the Project Assistant run simultaneously. Any changes that you make in one are reflected instantly in the other. For example, if you remove a feature while using the Installation Designer tab, that feature is no longer present in your installation project and does not appear in the Project Assistant.

Using the Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*

- *Transform*

When you create a new installation project, the Project Assistant is automatically displayed. The home page has an installation design diagram to help you visualize the steps that are involved in creating an installation. You can work within the Project Assistant to create your project or click the Installation Designer tab to further define a basic installation project.

Using the More Options, Other Places, and Help Links Sections in the Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The left column on each Project Assistant page contains one or more lists of links to help you in creating your installation and finding information:

- **More Options**—Provides additional configuration options relating to the specific area on the Project Assistant page. These are less common options that complete the functionality of the Project Assistant.
- **Other Places**—The view in the Installation Designer that corresponds to the current Project Assistant page. Clicking the link launches the full Installation Designer and activates that view.
- **Help Links**—This list provides links to help topics pertinent to the current Project Assistant page.

Navigating in the Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Task ***To navigate from one page of the Project Assistant to another, do one of the following:***

- To navigate directly to a specific page, click the appropriate icon in the navigation bar at the bottom of the page.
- To follow the Project Assistant steps sequentially, do one of the following:
  - Click the Next or Back arrow buttons to move forward or backward.

- Press CTRL+TAB to move to the next page and CTRL+SHIFT+TAB to move to the previous page.

- To move back to the Home page and view the installation design diagram, click the Home button on the navigation bar.

Opening the Installation Designer



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Installation Designer tab displays the views in the InstallShield Installation Development Environment (IDE). You can use this tab to add more complex and powerful elements to your installation project than you can with the Project Assistant. To open a view in the Installation Designer, click the Installation Designer tab.



Note *• The Installation Designer and the Project Assistant run simultaneously. Any changes that you make in one are reflected instantly in the other.*

Showing or Hiding the Project Assistant



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Project Assistant provides a simplified view of installation project tasks. You can perform all of the same tasks—and also further customize your project—through the Installation Designer.

If you prefer to create your InstallShield project entirely from within the Installation Designer, you can hide the Project Assistant so that its tab is not displayed in the InstallShield interface. Similarly, if the Project Assistant is hidden, you can choose to display it.



Task ***To show or hide the Project Assistant:***

On the **View** menu, click **Project Assistant**.

When the Project Assistant command has a check mark next to it, the tab for the Project Assistant is shown in the InstallShield interface. When the check mark is not displayed, the Project Assistant is hidden.

If the Project Assistant command does not have a check mark, the Installation Designer becomes the default tab whenever you create a new project.

Application Information Page



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Application Information page is where you specify general information about the application your project will install, including the application's name and version, your company's name and Web address, and the application icon.

Add or Remove Programs in the Control Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Add or Remove Programs in the Control Panel provides a list of the applications that are installed on a computer system. You can view information about particular programs and add, modify, or remove programs from Add or Remove Programs.

The information you provide in the Application Information page of the Project Assistant is used to populate information for your application's Add or Remove Programs information when your application is installed.

Company Name and Product Name in Your Installation



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

- *MSI Database*
- *Transform*

Your company name and product name are used in several places in your installation project.

How Your Company Name Is Used Throughout Your Installation Project

Your company name is used to set the default installation directory for your application. It is also used in Add or Remove Programs in the Control Panel for your application on the end user's system.

How Your Product Name Is Used Throughout the Installation Project

Your product name is used in your application's entry in Add or Remove Programs (in the support information link) on the target system. It is also used in setting the default installation directory.

Installation Requirements Page



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Installation Requirements page enables you to easily set installation requirements for the target system. For example, if your application requires a specific operating system in order to run properly, you can indicate that in the first section of this page.

Specifying Operating System Requirements in the Project Assistant



Project • *The ability to specify operating system requirements in the Project Assistant is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

When you specify operating system requirements on the Installation Requirements page of the Project Assistant, InstallShield creates launch conditions. These conditions are added to the **LaunchCondition** table of your .msi file; you can view and edit this table in the Direct Editor.

How InstallShield Creates the Operating System Launch Conditions

When you specify operating system requirements on the Installation Requirements page, you are essentially excluding operating systems that do not support your application.

For example, if you select only the check box for the latest Windows operating system, InstallShield creates a launch condition to exclude the operating systems that you did not select on the Installation Requirements page. With this type of launch condition, future versions of Windows operating systems are supported automatically because they are not excluded in the launch condition.

When Does the Installation Check for Requirements?



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

To ensure that the required software or operating system is present on the target system, the installation checks for these requirements in the beginning of the installation before any files are transferred.

Modifying the Run-Time Message for Software Requirements



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

If your installation has software requirements and the target system does not have the selected software, a run-time message is displayed during the installation. You can modify the message that is displayed.



Task ***To modify the run-time message:***

1. In the Project Assistant, open the **Installation Requirements** page.
2. Select **Yes** in answer to the software requirements question.
3. Select the software that your application requires. The default run-time message is displayed to the right.
4. Click the run-time message to edit it.

Creating Custom Installation Requirements



Project • *This information applies to the following project types:*

- *Basic MSI*

- *InstallScript MSI*
- *MSI Database*
- *Transform*

You can create customized installation requirements in the Installation Designer by using the product's Install Condition setting. For this setting, you can create conditions that the Windows Installer must evaluate before launching the installation. For more information, see Setting Product Conditions.

You can also use the System Search Wizard to search for a particular file, folder, registry key, or .ini value on the target system. The System Search Wizard enables you to create a condition with the search value.

Installation Architecture Page



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Installation Architecture page lets you specify the features you want your installation program to display to the end user. A feature is the smallest separately installable piece of your product from the end user's standpoint. Individual features are visible to end users when they select a Custom setup type during installation.



Note • *Features can contain subfeatures, subsubfeatures, and so forth, to as many levels as your installation program requires.*

Adding Features in the Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Task ***To add a feature:***

1. In the Project Assistant, open the **Installation Architecture** page.
2. Select **Yes** in answer to **Do you want to customize your Installation Architecture?**

3. To add a main feature, click the **Installation Architecture** explorer. To add a subfeature, click the feature that you want to be the parent feature, and then click **New**. The Project Assistant creates the new feature.
4. Name the feature or click **Rename** to name it later.

Determining Whether to Create a Multiple-Feature Installation



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Features are the building blocks of an installation, from the end user's perspective. Because of this, features should represent distinct and discrete pieces of functionality within your installation.

If your application has different blocks of functionality, you should create a multiple-feature installation. For example, if your installation contains your application (.exe file) and a help library (.hlp file), your installation project should contain at least two features—one for each piece of functionality.

To learn more about creating a multiple-feature installation within the Project Assistant, see Creating Installations with Multiple Features.

Creating Installations with Multiple Features



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

You can create an installation with multiple features using the Project Assistant or in the Installation Designer (IDE).



Task ***To create a multiple-feature installation in the Project Assistant:***

1. In the Project Assistant, open the **Installation Architecture** page.
2. Select **Yes** to indicate you want to customize your installation architecture.
3. Click the **Installation Architecture** explorer and then click **New**. The Project Assistant creates a new feature.
4. Press F2 or right-click the feature and select **Rename** to provide a name for the new feature.

5. To add another feature at the same level, click the **Installation Architecture** explorer and then click **New**. To create a subfeature, click the feature that you want to be the parent feature, and then click **New**. The Project Assistant creates the new feature.
6. Continue adding features and subfeatures as needed.

To learn about working with features in the Installation Designer, see Creating Features.

Default Features



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The *default feature* concept exists only in the Project Assistant. All resources (for example, files or registry data) that are added to an installation project need to be associated with a feature. If a resource is not associated with a feature, it is not installed to the target system at run time.

Using a default feature simplifies the authoring experience in the Project Assistant. You do not need to worry about associating project resources with a feature to ensure that they are installed. When you add registry data, create new shortcuts, or add files when All Application Data is selected, all of these resources are added to the default feature.

Setting the Default Feature

You can set the default feature in the Installation Architecture page of the Project Assistant.

What Happens If There Are No Features or No Default Feature Is Selected?

When you navigate to the Installation Architecture page or add data to the Application Files, Application Shortcuts, or Application Registry pages, InstallShield selects the first root feature as the default feature. If there are no features, InstallShield creates one silently.

Defining Feature Hierarchy



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Top-level features are the highest level in the feature hierarchy. Top-level features might include the application you want to install, a help library feature, and a sample projects feature.

Beneath the top-level features are subfeatures or *child* features. This is a feature that is dependent upon another feature for installation purposes. If the *parent* (or top-level) feature is not installed to the target system, the child feature is not installed.

Application Files Page



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Application Files page lets you specify the files you want to associate with each of your features.

Adding Files to Features in the Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Task ***To add a file to a feature:***

1. In the Project Assistant, open the **Application Files** page.
2. In the feature list at the top of the page, select the feature that should contain the file.
3. In **Destination Computer** explorer, select the folder to which you want to add the file.
4. Click **Add Files**. The **Open** dialog box opens.
5. Browse to the file that you want to add.
6. Click **Open** to add the file to the selected feature. The message "The file you have added ... may have dependencies" appears.
7. Click **Yes** if you want to have those dependencies automatically added to your installation project.

Removing Files from Features in the Project Assistant



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Task ***To remove a file from a feature:***

1. In the Project Assistant, open the **Application Files** page.
2. Click the file you want to remove and press **Delete**.

Adding Files to a Fixed Folder Location



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

If you know exactly where you want the installation to install your project files on the target system, you can hard-code a fixed folder destination.



Task ***To add files to a fixed folder location in the Project Assistant:***

1. In the Project Assistant, open the **Application Files** page.
2. Right-click **Destination Computer** and click **New Folder**.
3. For the new folder's name, type the drive in which the destination is located—for example, **C:**.
4. Beneath the drive letter folder, further define the destination path by adding subfolders.

Viewing Additional Predefined Folders



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *MSI Database*
- *Transform*

The Project Assistant's Application Files page displays the more commonly used predefined folders. You can view and hide predefined folders in this page.



Task ***To display additional predefined folders:***

1. In the Project Assistant, open the **Application Files** page.
2. Right-click **Destination Computer** and click **Show Predefined Folder**.
3. In the list of predefined folders, select the folder you want to display.



Tip • *To hide predefined folders, deselect them in the list of predefined folders.*

Application Shortcuts Page



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Application Shortcuts page lets you specify shortcuts for your application's files on the target system's desktop or Start menu. By default, this page displays a shortcut for each executable file that you have added to your project through the Project Assistant. You can delete these, and add shortcuts to other files that you have included in your installation project.

File Extensions



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

File name–extension associations, or file associations, are registry settings that tell Windows what application to use to open files of a certain type. For example, Windows typically opens text files (files with the .txt extension) with Windows Notepad, and opens bitmap files (files with the .bmp extension) with Microsoft Paint.

A file extension allows someone to identify the type of file without accessing the file. A suffix (**.abc**) is appended to the file name. The file extension is also useful in that another application can recognize whether the application can work with the file (for example, open the file or modify it), based on the extension.

In InstallShield, you can register your own file extensions by configuring the settings in the Advanced Settings area of a component in the Components view or the Setup Design view. Registering your file extension instructs the target machine's operating system to use your application to open files with your file extension when an end user clicks on a file.

Creating Shortcuts to Files That Are Not Included in the Installation



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

You can configure your installation to create a shortcut that points to a file that already exists on the target system. This file does not have to be included in your installation project.



Task ***To create a shortcut to a file that is not included in the installation:***

1. Open the Installation Designer.
2. In the View List under **System Configuration**, click **Shortcuts**.
3. In the **Shortcuts** explorer, right-click the destination directory that you want to contain the shortcut and then click **New Shortcut to Preexisting File**. The **Browse for Shortcut Target** dialog opens.
4. Browse to the target file's location and enter the file's name in the **File Name** setting.
5. Click **OK**.
6. Configure the shortcut's settings.

Modifying a Default Shortcut in the Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Task ***To modify a default shortcut:***

1. In the **Project Assistant**, open the **Application Shortcuts** page.
2. Select the shortcut that you want to modify.
3. Make the required changes as needed.

Associating a Shortcut's Target with a File Extension in the Project Assistant

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

You can associate a shortcut's target with a file extension. When you do this, Windows will use the target file to launch files with the specified file extension. For example, if you enter .txt, when the end user opens a .txt file, the target file of this shortcut is launched to open the .txt file.

Task ***To associate a shortcut's target with a file extension:***

1. In the **Project Assistant**, open the **Application Shortcuts** page.
2. Click the shortcut to activate the shortcut options.
3. Select the **Associate shortcut with file extension** option.
4. Type the file extension that you want to associate with this shortcut's target—for example, **txt**. You can add multiple extensions by separating them with a comma.

Application Registry Page

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Application Registry page lets you specify any registry data that your application requires.

Updating the Registry



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The registry is a database for your computer's configuration information. Information included in a computer's registry includes user profiles, hardware and software installed on the computer, and property settings.

How Do I Know What Registry Data My Application Requires?

The application developer should be able to provide registry information for you. Specifically, you will need to know if the application you are installing requires any user-specific (HKEY_CURRENT_USER) or machine-specific (HKEY_LOCAL_MACHINE) settings.

The developer can provide a .reg file that you add to your installation. InstallShield allows you to import .reg files into your installation project.

Configuring Registry Data in the Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Task ***To configure registry data:***

1. In the **Project Assistant**, open the **Application Registry** page.
2. Select **Yes** in answer to the question about configuring registry data.
3. Right-click the registry item to which you want to add the data, select **New**, and point to **Key**.
4. Name the key.
5. Right-click the key, select **New**, and point to the appropriate command. You can pick Default Value, String Value, Binary Value, DWORD Value, Multi-String Value, or Expandable String Value, depending on the type of data you want to register.

Modifying Registry Data Values in the Project Assistant



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Task ***To modify registry data:***

1. In the **Project Assistant**, open the **Application Registry** page.
2. Double-click the data.

   If you double-clicked a multi-line string, the **Multi-Line String Value** dialog box opens. If you double-clicked any other type of registry data, the **Edit Data** dialog box opens.
3. Edit the data and click **OK**.

Associating Registry Data with Features



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

All the registry data that you add in the Project Assistant's Application Registry page is added to your project's default feature. You can associate your registry data with another feature in the Installation Designer.



Task ***To use the Installation Designer to associate registry data with a feature other than the default feature:***

1. In the View List under **System Configuration**, click **Registry**.
2. In the **View Filter** list, select the feature with which you want to associate the registry data.
3. Create or drag and drop the registry data in the appropriate registry location.

Using Variable Data Types in Registry Data



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

InstallShield allows you to use variable data types or properties when creating registry data for your installation project.

Task ***To use INSTALLDIR as a variable in the registry (Windows Installer-based projects only):***

1. In the **Project Assistant**, open the **Application Registry** page.
2. Select **Yes** to indicate that you want to configure the registry data that your application will install.
3. Right-click **HKEY_CLASSES_ROOT**, point to **New**, and select **Key**.
4. Name the key **Installation Location**.
5. Right-click the **Installation Location** key, point to **New**, and select **String Value.**
6. Name the string value **My Installation Location**.
7. Double-click the **My Installation Location** key. The **Edit Data** dialog box opens.
8. In the **Value Data** field, type **[INSTALLDIR]**.

At run time, the value of **[INSTALLDIR]** is replaced with the installation directory.

Application Paths

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The application path registry key contains data that Windows uses as a private search path for the specified application's .dll files. If you install an application's .dll files into a directory not found in the PATH environment variable (and not into the application's directory), you should set the appropriate application path to include the .dll file directory during installation. Application path information is stored in the registry under HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\AppName.exe.

Installation Interview Page

Project *• This information applies to the following project types:*

- *Basic MSI*

- *InstallScript*
- *MSI Database*

The Installation Interview page lets you specify the dialogs you want the end user to see when your installation program runs. Based on your answers to the questions on this page, the Project Assistant adds the corresponding dialog to your installation project.

Specifying Dialogs for Your Installation in the Project Assistant



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *MSI Database*

Note that the option for launching your application when the installation completes is not available in InstallScript projects.

The questions that are displayed on the Installation Interview page of the Project Assistant help you specify which dialogs you want the end user to see when your installation program runs.



Task ***To specify dialogs for your installation:***

1. **Do you want to display a License Agreement Dialog?—**Select **Yes** to browse to your license agreement file.
2. **Do you want to prompt users to enter their company name and user name?—**Select **Yes** to display a dialog requesting this information.
3. **Do you want your users to be prompted to modify the installation location of your application?**—Select **Yes** to present a dialog that allows end users to change the installation location. See Allowing End Users to Modify the Installation Location for more information.
4. **Do you want users to be able to selectively install only certain parts of your application?—**Select **Yes** to allow end users to select which parts of the application they want to install. See Creating Selectively Installable Installations for more information.
5. **Do you want to give users the option to launch your application when the installation completes?—** Select **Yes** and browse to your application file. When this option is set to **Yes**, the final dialog in the installation presents a check box that allows end users to immediately launch your application upon clicking the **Finish** button.



Tip *• To add a custom graphic to your installation dialogs, click the link in the More Options section of this page to launch the Dialog Images dialog box.*

Allowing End Users to Modify the Installation Location

If you want to provide end users with control over where your software is installed on their system, you can allow them to modify the installation location.

In a Windows Installer-based installation, the Windows Installer property **INSTALLDIR** serves as the default installation directory. When you allow users to modify the installation location, the CustomSetup dialog is presented during the installation.

In an InstallScript project, TARGETDIR serves as the default installation directory. When you allow users to modify the installation location, the SdSetupType2 dialog is presented during the installation.

License Agreements



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *MSI Database*

In order to install your product, you may require that your end users agree to certain legal requirements. For example, most software vendors do not allow end users to copy or distribute their software to others.

Your installation can present an End User License Agreement (EULA) in the License Agreement dialog at run time. The EULA is a legal contract between you and the end user, with regard to the use of your product.

The License Agreement dialog displays your license agreement text and contains a radio button group (Yes or No). If the end user does not agree to accept the EULA, your software is not installed and the installation ends.



Task ***To add a License Agreement dialog to your project in the Project Assistant:***

1. In the **Project Assistant**, open the **Installation Interview** page.
2. Select **Yes** to indicate that you want to add a **License Agreement** dialog.
3. Type the path to your license agreement file or browse to the file.



Project *• For Basic MSI and MSI Database projects, the file must be a rich text file (.rtf).*

For InstallScript projects, the file type depends on which license dialog that you are using and which parameters you are passing to it. The SdLicenseEx and SdLicense2Ex functions have support for .rtf files and text files (.txt).

Creating Selectively Installable Installations

You can allow your end users to select which portions of your installation they want to install to their systems. This is a custom installation, which provides a list of the available features within your installation. The end user can select the features to install in a dialog presented at run time.

For example, your installation might contain your application's executable (.exe) file, a documentation (.chm) file, and a samples file. All of these files are contained in different features, which are provided in an option list to the end user. If the end user needs only the application, they can select to install the executable file, but not the documentation or samples file.

Installation Localization Page

Edition • *Multilingual language support is available in the Premier edition of InstallShield.*

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Installation Localization page lets you specify the languages your installation supports, and specify string values and associated identifiers, which you can use in your end user interface to make your installation more easily localizable in other languages.

Note • *You can specify the set of string data that you want to edit in the drop-down menu at the top of the page.*

Localizing String Data from the Project Assistant

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Much of the text in your installation project is stored as string entries. To make localizing your installation easier, InstallShield provides the capability to export your strings for translation and import the translated strings back into your project.

Task ***To export all of the string data in your project for translation:***

1. In the **Project Assistant**, open the **Installation Localization** page.
2. In the data list at the top of the page, select **All String Data**.
3. In the list of languages, select a language.
4. In the **More Options** area, click **Export localizable strings**. The **File Name** dialog box opens.
5. Save the text (.txt) file to the location you want.
6. Send the text file to your translation company or department.

Task ***After the strings in your text file are translated, you can import the strings back into your project:***

1. In the **Project Assistant**, open the **Installation Localization** page.
2. In the data list at the top of the page, select **All String Data**.
3. In the list of languages, click the language that matches the strings that you want to import.
4. In the **More Options** area, click **Import localizable strings**. The **File Name** dialog box opens.
5. Browse to your .txt file and click **Open**. The translated strings are imported back into your project.

How Localized String Data Is Used in the Installation

There are two types of string data in an installation project—string data that is specific to your installation and string data that is specific to your application.

The majority of the string data in an InstallShield project is used at run time when an end user runs the installation. However, some of the string data is installed to the target system. For example, a localized shortcut is localized string data that is installed to the end user's system.

Build Installation Page

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Tip *• The following information applies to releases that are built within InstallShield (without integration with Visual Studio). For information on building an InstallShield release from within Visual Studio, see Building Releases in Microsoft Visual Studio.*

The Build Installation page lets you specify what type of distribution you want to build and, optionally, the location to which you want to copy the distribution files. It also enables you to digitally sign the package.

Building Your Installation from the Project Assistant

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Tip *• The following information applies to releases that are built within InstallShield (without integration with Visual Studio). For information on building an InstallShield release from within Visual Studio, see Building Releases in Microsoft Visual Studio.*

Task ***To build your installation:***

1. In the **Project Assistant**, open the **Build Installation** page.
2. Select the installation image types you want to build.

   Note *• The* ***Single MSI Package*** *option should be selected for machines that already have Windows Installer installed them since no attempt will be made to check and install the engine if it is not there.*

3. If you want InstallShield to automatically copy your installation to another location after the build finishes, click the **Optional distribution settings** link for each build option and specify the location.

   If you want InstallShield to distribute your installation after the build finishes, click the **Optional distribution settings** link for each build option and select the **Distribute After Build** check box.

4. To digitally sign your **Setup.exe** file to assure your end users that the code within your application has not been modified or corrupted, click the **Digitally Sign Setup hyperlink**. The **Digitally Sign Setup** dialog box opens. Configure the settings as needed.
5. Click **Build Installations**.

The Output window opens, and the Build tab displays information about the progress of the build. The build is finished when the Build tab displays the log file information.

Tip *• The Signing tab in the Installation Designer lets you specify which portions of your installation should be digitally signed at build time. InstallShield enables you to sign any and all of the following files in a release, depending on what type of project you are using:*

- *Windows Installer package (.msi file or .msm file) for Basic MSI, InstallScript MSI, and Merge Module projects*
- **Setup.exe** *file for Basic MSI and InstallScript MSI projects*
- *Media header file for InstallScript projects*
- *Package (self-extracting single-executable file) for InstallScript projects*
- *Any files in your release, including your application files*

To learn more, see Digitally Signing a Release and Its Files at Build Time.

Windows Logo *• All executable files (including .exe, .dll, .ocx, .sys, .cpl, .drv, and .scr files) in an installation must be digitally signed for the Windows logo program.*

After Completing the Project Assistant: Next Steps



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

After going through the Project Assistant pages and completing the fields, you have an installation project framework that you can use as a functional installation or you can further customize to fit your needs.

Further Customizing Your Project

The Installation Designer provides an easy way to access all of the installation creation views available in InstallShield. Click the Installation Designer tab at the top of the Project Assistant workspace to display the views.

Within the Installation Designer, you can see the View List, which is a list of all available views for the project type that you are creating. To display the View List on the left side of the workspace, press F4.

Learning About InstallShield Views

For detailed information about each of the InstallShield views, refer to the topics in the View List section of the InstallShield Help Library.

Working with the InstallShield Interface

The InstallShield interface is a graphical user interface with conventional Windows-based elements such as a menu bar, a toolbar, and dialog boxes. This section includes topics that explain how to perform basic tasks using these elements and how to customize the interface.

Displaying the View List



Task ***To see the View List in the InstallShield interface:***

1. Click the **Installation Designer** tab at the top of the InstallShield interface.
2. Do one of the following to display the View List:
   - On the **View** menu, click **View List**. The **View List** appears on the left side of the InstallShield interface.
   - Click the toolbar's **View List** button.
   - Press F4 to hide or show the **View List**.

Opening Views in the InstallShield User Interface

The first step in many InstallShield procedures is to open a particular view in the Installation Development Environment (IDE).



Task ***To open a view:***

1. Click the **Installation Designer** tab. The **View List** is displayed along the left side of the IDE. If the **View List** is not displayed, see Displaying the View List.
2. In the **View List**, select the view you want to open. To see all available views, expand the View List folders.

Working with the Group Box Area in Various Views

A number of views in InstallShield have a group box area that lets you organize the rows in the view. The views that contain this group box support multiple levels of grouping simply by dragging the column headers and dropping them onto the group box (the area that says, "Drag a column header here to group that column"). InstallShield displays the rows in the view hierarchically according to column arrangement in the group box. Examples of views that contain the group box are the String Editor, Property Manager, the Redistributables, Prerequisites, and Path Variables views.

Note the following tips for working with the group box:

- To move a column header to the group box area, drag the column header and drop it onto the group box.
- To copy a column header onto the group box area, press CTRL while dragging the column header and dropping it onto the group box. When you do this, the column header is displayed as a column header, and in the group box.
- When you are dropping group box headers onto the group box area, you can drop them onto other column headers. This enables you to organize rows hierarchically.
- To remove a column header from the group box, drag it from the group box area and drop it onto the row of column headers. When you are dragging it over the row of column headers, InstallShield displays arrows to indicate where in the row the column header would be displayed if you dropped it.
- To sort the items in the grid by a particular column header, click the column header in the group box or in the row of column headers.

The following examples demonstrate different ways for using the group box to organize content in views.

Default Behavior: Empty Group Box Area

By default, no column headers are displayed in the group box. The following screen shot shows part of the Redistributables view. Items are sorted by the Name column.

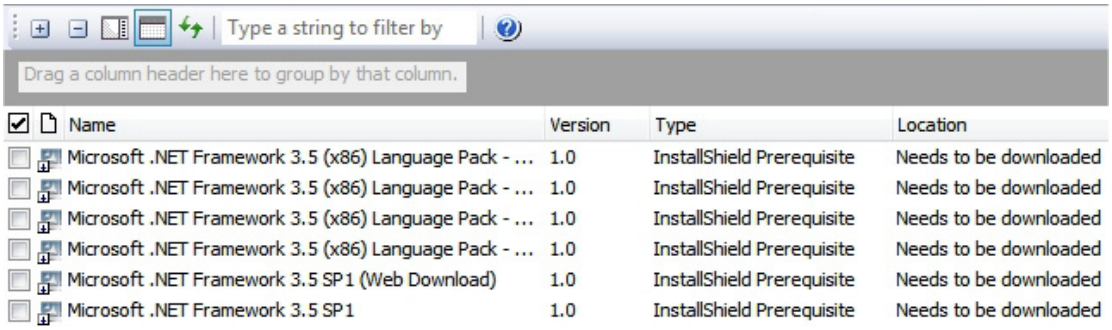


Figure 2-3: Empty Group Box Area

Grouping by One Column Header

If you press CTRL while dragging and dropping one column header onto the group box, InstallShield arranges the rows in the grid into groups of items. The following screen shot shows part of the String Editor view. Rows are organized to help identify the recently added and modified strings that need to be translated.

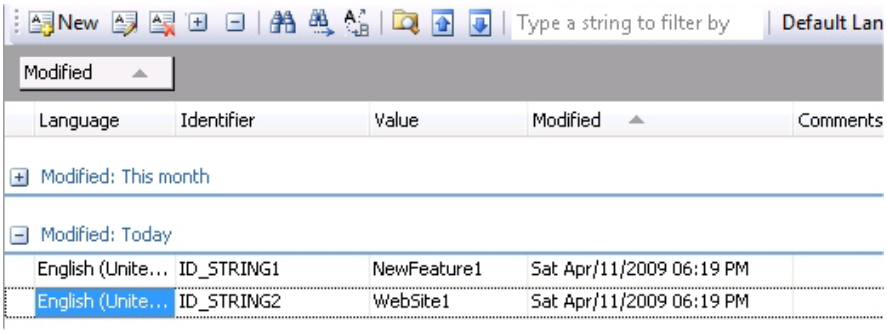


Figure 2-4: Grouping Rows by One Column Header

Grouping by Two Column Headers

If you press CTRL while dragging and dropping one column header onto another column header in the group box, InstallShield arranges the rows in the grid into multiple groups of items. The following screen shot shows part of the Redistributables view. Rows are organized to help identify the InstallShield prerequisites and merge modules that have been added to the project.

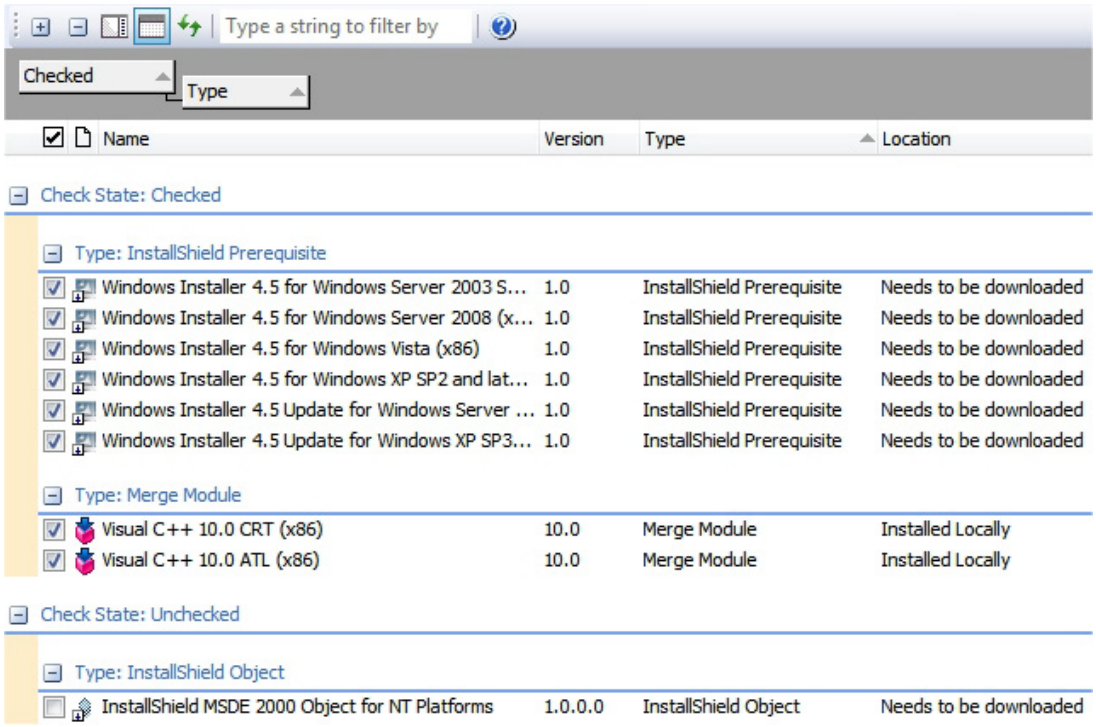


Figure 2-5: Sorting Rows by the Check Box Column and Then the Type Column

Showing or Hiding Toolbars

Task ***To show or hide a toolbar, do one of the following:***

- Right-click a toolbar and select the toolbar that you want to be displayed or hidden.
- On the **Tools** menu, click **Customize**. The **Customize** dialog box opens. Select the check box for each toolbar that you want to be displayed. Clear the check box for each toolbar that you want to be hidden.

Adding Buttons and Menus to a Toolbar

Task ***To add a button or menu to a toolbar:***

1. Ensure that the toolbar that you want to change is visible.
2. On the **Tools** menu, click **Customize**. The **Customize** dialog box opens.
3. Click the **Commands** tab.
4. In the **Categories** box, click the category for the button or menu that you want to add.
5. Drag the button or menu from the **Commands** box to the appropriate toolbar.



Tip • *To create your own custom toolbar, drag the button or menu to the empty gray area near the toolbars.*

Removing Buttons and Menus from a Toolbar



Task ***To remove a button or menu from a toolbar:***

1. Ensure that the toolbar that you want to change is visible.
2. On the **Tools** menu, click **Customize**. The **Customize** dialog box opens.
3. Right-click the button or menu that you want to remove, and then click **Delete**.

Creating Custom Toolbars



Task ***To create a custom toolbar:***

1. On the **Tools** menu, click **Customize**. The **Customize** dialog box opens.
2. Click the **Toolbars** tab.
3. Click the **New** button. The **New Toolbar** dialog box opens.
4. In the **Toolbar name** box, enter a descriptive name for the toolbar, and click **OK**.
5. Customize the new toolbar by adding menus or buttons.

Docking or Undocking the Output Window

The Output window or its individual tabs can be docked to any side of the workspace in InstallShield, or they can be dragged to free-floating positions.

If you drag the Output window or one of its tabs to the edge of the InstallShield interface, it becomes a docked window. If you drag the Output window or one of its tabs away from any of the edges of the InstallShield interface, it becomes undocked.



Task ***To undock the Output window:***

Drag the title bar of the Output window to the new location. Resize the Output window as needed.



Task ***To dock the Output window:***

Drag the title bar of the Output window to the left, right, top, or bottom edge of the InstallShield interface.

Task ***To undock a tab on the Output window:***

Drag the tab to the new location. Resize the Output window as needed.

Task ***To dock one of the Output window tabs:***

Drag the tab to the left, right, top, or bottom edge of the InstallShield interface.

Working with the Script Editor Pane in Various Views

Several views in InstallShield have a script editor pane that you can use to write code for your projects. Examples of views that contain the script editor pane are:

- **InstallScript view**—When you select a script file in this view, InstallShield displays a script editor pane where you can write InstallScript code.
- **SQL Scripts view**—When you select a SQL script in this view, the Script tab shows a script editor pane that lets you edit your SQL file.
- **Custom Actions and Sequences view**—When you select a VBScript or JScript custom action in this view, the Script tab shows a script editor pane that lets you edit your VBScript or JScript file.

This section of the documentation describes how to use the script editors in InstallShield.

Using Bookmarks in the Script Editors

Bookmarks are markers that let you jump to specific lines within your script with a minimum number of keystrokes. Bookmarks are visible in the left margin of the script editor. Note that InstallShield does not save bookmarks when you close your project.

Task ***To add a new bookmark or clear an existing bookmark:***

1. Do one of the following:
   - Place the insertion point in the line of your script that you want to contain a bookmark.
   - Place the insertion point in the line of your script that contains the bookmark that you want to
2. Press ALT+K.

Task ***To move the insertion point to the next line that contains a bookmark:***

Press CTRL+K.

Task ***To move the insertion point to the previous line that contains a bookmark:***

Press CTRL+SHIFT+K.

Enabling or Disabling Auto Completion in the Script Editors

InstallShield lets you specify whether you want to be able to use automatic completion when you are typing in the script editor in various views (for example, the InstallScript view and the SQL Scripts view). If auto completion is enabled, InstallShield displays a pop-up list of alphabetically ordered functions, keywords, constants, and string identifiers that begin with the letters that you are typing in the script editor. Instead of manually typing the entire word, you can select it in the pop-up list, and InstallShield adds it to your script.

Auto completion can increase your efficiency because it can reduce the time that you spend typing code. It can also help you avoid typographical errors in your code.

Tip *• Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same auto completion settings are used for all script editors. If you change the settings in the SQL Scripts view, the settings are also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*

Task ***To specify whether auto completion should be available in the script editors:***

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Auto completion** setting, select the appropriate option:
   - **Yes**—InstallShield displays a pop-up list of available functions, keywords, constants, and string identifiers that begin with the letters that you are typing code in the script editors. This is the default option.
   - **No**—InstallShield does not display a pop-up list when you are typing code in the script editors.
3. In the **Include local variables** setting, select the appropriate option:
   - **Yes**—InstallShield adds local variables that are defined in InstallScript to the pop-up list of available functions and other script words that begin with the letters that you are typing code in the InstallScript editor. This is the default option.
   - **No**—InstallShield does not include local variables in the pop-up list when you are typing code in the InstallScript editor.

     Note that if you select No, the Functions, Properties, and Methods folders in the center pane of the InstallScript view do not list any functions, properties, or methods from your script files.

     If Yes is selected for this setting and you notice performance issues when you are typing code in the InstallScript view, you might want to change this setting to No.

Note *• The* ***Include local variables*** *setting applies to the script editor in the InstallScript view; it does not have any effect on the script editors in other views, such as the SQL Scripts view. This setting is ignored if you select* ***No*** *for the* ***Auto completion*** *setting.*

4. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield enables or disables auto completion in the InstallScript editor as needed.

For details about how auto completion works, see Using Auto Completion when Writing Code in the Script Editors.

Using Auto Completion when Writing Code in the Script Editors

If auto completion is enabled for the script editors in InstallShield, InstallShield displays a pop-up list of alphabetically ordered functions, keywords, constants, and string identifiers that begin with the letters that you are typing in the script editor. The pop-up list contains language-appropriate options, depending on which view you are using. For example, the script editor in the InstallScript view supports auto completion of InstallScript code. The VBScript script editor in the Custom Actions and Sequences view supports auto completion of VBScript.

In addition, if auto completion for local variables is enabled, the pop-up list in the script editor of the InstallScript view also contains local variables.

To learn how to enable auto completion, see Enabling or Disabling Auto Completion in the Script Editors.



Task ***To use auto completion for a function name, a keyword, a constant, or string identifier:***

1. Open the view that contains the appropriate script editor, and select the script that you want to edit.
2. Place the insertion point where you want to enter code in your script.
3. Do one of the following:
   - Type the first characters of the function name, keyword, or other script word.

     Depending on the view that you are using and the language that is entered in the script editor in that view, InstallShield displays a pop-up list of alphabetically ordered functions, keywords, and other script words that begin with the letters that you are typing in the script editor. The first word that matches the characters that you typed is selected in the list. If no word matches the characters that you typed, no word in the list is selected.
   - If you are using the script editor in the InstallScript view and you want to use auto completion for a list of available string identifiers, enter the string constant operator (@). InstallShield displays a pop-up list of alphabetically ordered string identifiers.
4. If the selected word is not the one you want (or no word is selected), select another word in one of the following ways:
   - To navigate through the list, use the pop-up list's scroll bar; then, click the desired word to select it.
   - To change the selection to the previous or next word, use the UP ARROW and DOWN ARROW keys.
5. To paste the selected word into your script, double-click the word, or press the ENTER key or the TAB key.

To close the list without pasting the selected word, press ESC or click outside the pop-up list.

Enabling or Disabling InstallScript Function Call Tips in the Script Editor

InstallShield lets you specify whether you want InstallShield to display an InstallScript function call tip—a type of tooltip—when you are entering a function call in your script in the InstallScript view. A function call tip shows a built-in function's parameter information. It also shows a description of the built-in function, as well as a description of the parameter that you are entering.

Function call tips let you see language information within the script editor, without requiring you to switch to a different window.



***Tip** • Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same auto completion settings are used for all script editors. If you change the settings in the SQL Scripts view, the settings are also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*



Task ***To specify whether function call tips should be shown in the script editor of the InstallScript view:***

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Show function call tips** setting, select the appropriate option:
   - **Yes**—InstallShield displays function call tips when you are entering function calls in your script. This is the default option.
   - **No**—InstallShield does not display function call tips when you are entering function calls in your script.
3. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield enables or disables function call tips in the InstallScript editor as needed.

For details about how function call tips work, see Viewing Function Call Tips for an InstallScript Function in the Script Editor.

Viewing Function Call Tips for an InstallScript Function in the Script Editor

If function call tips are enabled, InstallShield shows InstallScript function call tips—a type of tooltip—when you are entering function calls in your script in the InstallScript view. A function call tip shows a built-in function's parameter information. It also shows a description of the built-in function, as well as a description of the parameter that you are entering.

To learn how to enable function call tips, see Enabling or Disabling InstallScript Function Call Tips in the Script Editor.



Task ***To view a description of a function, as well as parameter details about that function:***

1. Enter a function name in your script by typing it or using automatic function name completion, as described earlier.
2. Type a left parenthesis—**(**—after the function name. InstallShield displays a function call tip that shows a complete call to the function, including all of its parameters. The tip also includes a description of the function, as well as a description of the next parameter that you need to enter. The first parameter appears in blue, by default.

3. Type the parameters as indicated by the call tip. Each time that you type a comma, the next parameter in the function syntax appears in blue, by default.
4. To close the call tip, type the right parenthesis—**)**. Or, press ESC or click in the script editor pane outside the call tip.

Tip *• To view additional details about a particular built-in function, constant, or other InstallScript word, place the insertion point within that word, and then press F1. The InstallShield Help Library opens, enabling you to learn more about that particular word.*

Enabling or Disabling Syntax Highlighting in the Script Editors

InstallShield lets you specify whether you want the script editors in various views in InstallShield to display script files with color attributes that identify keywords, comments, strings, and other script elements. Each category is displayed in a different color so that you can easily identify them.

This functionality—called *syntax highlighting*—helps to improve the readability and context of code. In addition, it can help you avoid errors that are caused, for example, when you attempt to use a reserved word as a user-defined identifier. It can also help you locate other errors in your script, such as misspelled keywords, missing quotation marks at the end of a string, and missing comment characters.

Tip *• Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same syntax highlighting and color settings are used for all script editors. If you change the settings in the SQL Scripts view, the settings are also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*

Task ***To specify whether syntax highlighting should be enabled in the script editors:***

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Syntax highlighting** setting, select the appropriate option:
   - **Yes**—InstallShield uses the syntax highlighting that is defined in the **Colors** area of the **Script Editor Properties** dialog box in all of the script editors. This is the default option.
   - **No**—InstallShield does not use any syntax highlighting in any of the script editors.
3. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield enables or disables syntax highlighting in the script editors as needed.

InstallShield lets you define the foreground color and background color that is used for each script word category. To learn how, see Changing Colors for Syntax Highlighting in the Script Editors.

Changing Colors for Syntax Highlighting in the Script Editors

InstallShield lets you change the colors that are used to differentiate among different script elements in the script editor in various views (for example, the InstallScript view and the SQL Scripts view). You can change the foreground color (the color that is used for the text), as well as the background color, for script elements such as comments, functions, and strings.

This functionality—called *syntax highlighting*—helps to improve the readability and context of code. In addition, it can help you avoid errors caused, for example, when you attempt to use a reserved word as a user-defined identifier. It can also help you locate other errors in your script, such as misspelled keywords, missing quotation marks at the end of a string, and missing comment characters.

Tip *• Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same syntax coloring settings are used for all script editors. If you change the settings in the SQL Scripts view, the settings are also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*

Task ***To change the syntax highlighting that is used in the script editors:***

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Colors** area, click the setting for the script element that you want to configure. InstallShield displays foreground and background buttons in the setting.
3. Do one of the following:
   - Click the foreground button or the background button. The **Colors** dialog box opens, enabling you to select the color that you want to use.
   - Enter the RGB values for the color that you want to use for the foreground or the background. For example, the entry **0; 0; 0 / 255; 255; 255** indicates that InstallShield should use black (which has an RGB value of 0; 0; 0) for the foreground and white (255; 255; 255) for the background. If you do not want to set a color for a particular area, you can use dashes (-;-;-).
4. Click **OK**. The **Script Editor Properties** dialog box closes.

If script highlighting is enabled, InstallShield changes the colors that are used to display text in the script editor as needed. To learn more, see Enabling or Disabling Syntax Highlighting in the Script Editors.

To make the best use of this functionality, note the following:

- Misspelled reserved words are not recognized by the editor and are not displayed with syntax coloring. If you see in your script a "reserved word" that is not displayed with indicating attributes, it is possible that it is misspelled.
- Coloring of string literals includes both the opening and closing quotation marks. If the closing quotation mark is missing, string coloring will extend to the end of the line; in that case, text that should have followed the quotation will be displayed as though it were part of the string literal.
- Syntax coloring makes it easy to identify comments that open with **/*** and are not closed properly with ***/**. In that case, all of the text that follows the comment is displayed with the comment color attribute.
- In lines that include comments starting with two slashes (**//**), all text from the comment character to the end of the line is recognized as a comment.

- In the script editor that is displayed in the InstallScript view, InstallShield does not use syntax coloring for files that have an extension other than .rul or .h (for example, log or report files).

Changing the Font that Is Used to Display Text in the Script Editors

InstallShield lets you change the font that is used in the script editor in various views (for example, the InstallScript view and the SQL Scripts view).

Tip *• Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same font is used for all script editors. If you change the font in the SQL Scripts view, the font is also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*

Task

To change the font, font style, or font size that is used to display text in the script editors:

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **General** area, click the **Font** setting, and then click the ellipsis button (**...**) in this setting. The **Font** dialog box opens.
3. Specify the font changes as needed.
4. Click **OK**. The **Font** dialog box closes.
5. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield changes the font that is used to display text in the script editor as needed.

Showing or Hiding Line Numbering in the Script Editors

InstallShield lets you specify whether you want the left margin of the script editors in InstallShield to contain line numbers.

Tip *• Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same line numbering setting is used for all script editors. If you change the setting in the SQL Scripts view, the settings are also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*

Task

To specify whether line numbering should be enabled in the script editors:

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Line numbering** setting, select the appropriate option:
   - **Yes**—InstallShield adds line numbers to the left margin of all of the script editors.
   - **No**—InstallShield does not show any line numbers in the left margin of any of the script editors. This is the default option.
3. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield shows or hides line numbering in the script editors as needed.

Going to a Line Number in a Script that Is Displayed in a Script Editor



Task ***To move the insertion point to a specific line number in your script:***

1. On the **Edit** menu, click **Go To**. The **Go To Line** dialog box opens.
2. In the **Line** box, type the number of the line where you want to move the insertion point.
3. Click **OK**.

Enabling or Disabling Automatic Indentation in the Script Editors

InstallShield lets you specify whether you want to use automatic indentation in the script editors when you are writing code. Auto indentation may help improve readability.

If automatic indentation is enabled, InstallShield indents a new line according to the indentation that is used in the previous line. When you press ENTER in the script editor pane, the insertion point appears on a new line, positioned directly beneath the first character of the previous line. To stop indenting a new line, press BACKSPACE; doing so removes the indentation.

If automatic indention is disabled, a new line is not indented—that is, the insertion point is placed in the first column of the new line.



Tip *• Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same automatic indentation setting is used for all script editors. If you change the setting in the SQL Scripts view, the settings are also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*



Task ***To specify whether automatic indentation should be enabled in the script editors:***

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Automatic indentation** setting, select the appropriate option:
   - **Yes**—InstallShield adds support for automatic indentation in all of the script editors. This is the default option.
   - **No**—InstallShield does not use automatic indentation in any of the script editors.
3. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield enables or disables automatic indentation in the script editors as needed.

Setting the Tab Width for the Script Editors

InstallShield lets you specify the width of tabs in the script editor in various views (for example, the InstallScript view and the SQL Scripts view).

Tip *• Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same tab setting is used for all script editors. If you change the setting in the SQL Scripts view, the settings are also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*

Task ***To specify the width that InstallShield should use for tabs that are entered in the script editors:***

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Tab width** setting, enter the width—as a multiple of the character space size—that you want for tabs in your scripts. The default value is **4**.
3. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield adjusts the widths of the tabs in your script as needed.

Enabling or Disabling Syntax Folding in the Script Editors

InstallShield lets you specify whether you want the script editors in various views in InstallShield to include support for syntax folding. When syntax folding is enabled, InstallShield adds a plus sign (+) or a minus sign (–) in the margin next to each line of code that starts an expandable or collapsible block of script. You can click a plus sign to expand hidden code, or a minus sign to hide code.

Syntax folding can help you minimize the clutter of large scripts and focus on the code that is relevant to the work that you are currently doing. It can also help you see the overall structure of a script.

Tip *• Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same syntax folding setting is used for all script editors. If you change the setting in the SQL Scripts view, the setting is also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*

Task ***To specify whether syntax folding should be enabled in the script editors:***

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Syntax folding** setting, select the appropriate option:
   - **Yes**—InstallShield uses the syntax folding in all of the script editors.
   - **No**—InstallShield does not use any syntax folding in any of the script editors. This is the default option.
3. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield enables or disables syntax folding in the script editors as needed.

Showing or Hiding Whitespace in the Script Editors

InstallShield lets you specify whether you want the script editors in various views in InstallShield to display a symbol or other mark in place of each instance of whitespace in your script. For example, if you select Yes, InstallShield displays each space character in the script as a middle dot (·) and each tab character as an arrow. InstallShield also identifies line breaks when whitespace is enabled.



***Tip** • Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same whitespace setting is used for all script editors. If you change the setting in the SQL Scripts view, the setting is also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*



Task ***To specify whether whitespace should be shown or hidden in the script editors:***

1. Right-click in the script editor pane and click **Properties**. The **Script Editor Properties** dialog box opens.
2. In the **Show whitespace** setting, select the appropriate option:
   - **Yes**—InstallShield shows whitespace characters and symbols in all of the script editors.
   - **No**—InstallShield does not show whitespace characters and symbols in any of the script editors. This is the default option.
3. Click **OK**. The **Script Editor Properties** dialog box closes.

InstallShield shows or hides whitespace characters and symbols in the script editors as needed.

Dragging and Dropping Text in the Script Editors

The script editors in InstallShield support drag-and-drop text editing, enabling you to move or copy selected text from one location in your code to another.

Moving Text



Task ***To move text:***

1. Select the text that you want to move.
2. Place the mouse pointer over the selected text.
3. Press and hold down the left mouse button.
4. When you see a small rectangle appear beneath the mouse pointer, move the pointer to the location in your script to which you want to move the selected text. Continue to hold down the left mouse button while you move the pointer.
5. When the pointer is at the location in which you want to move the text, release the left mouse button.

Copying Text

Task ***To copy selected text using the drag and drop method:***

1. Select the text that you want to move.
2. Place the mouse pointer over the selected text.
3. Press CTRL, and then press and hold down the left mouse button.
4. When you see a small rectangle and a plus sign beneath the mouse pointer, move the pointer to the location in your script to which you want to copy the selected text. Continue to hold down CTRL and the left mouse button while you move the pointer.
5. When the pointer is at the location in which you want to move the text, release the left mouse button.

Printing a Script File from Within a Script Editor

InstallShield lets you print a script that is displayed in the script editors in various views (for example, the InstallScript view and the SQL Scripts view).

Task ***To print the script that is displayed in the active script editor pane:***

1. On the **File** menu, click **Print**. Note that this command is enabled only if the insertion point is in the script editor pane. The **Print** dialog box opens.
2. Specify the options that you prefer.
3. Click **OK**.

Keyboard Shortcuts for the Script Editors

The script editors in InstallShield include support for the following keyboard shortcuts.

Table 2-8 • Keyboard Shortcuts for the Script Editors

Keyboard Shortcut	Description
CTRL+C CTRL+INSERT	Copies the selected text to the Clipboard.
CTRL+X SHIFT+DELETE	Deletes the selected text and puts it in the Clipboard.
CTRL+F	Finds the specified text.
CTRL+H	Displays the Replace dialog box, which lets you search for text or other characters and replace them.

Table 2-8 • Keyboard Shortcuts for the Script Editors (cont.)

Keyboard Shortcut	Description
CTRL+G	Moves to a specified line.
TAB	Indents the selected text to the right one tab stop.
CTRL+U	Changes the selected text to all lowercase.
CTRL+V SHIFT+INSERT	Inserts the Clipboard contents at the insertion point.
CTRL+Y	Redoes the previously undone action.
CTRL+A	Selects all of the text in the entire document.
CTRL+Z ALT+BACKSPACE	Undoes the last action.
SHIFT+TAB	Outdents the selected text.
CTRL+SHIFT+U	Changes the selected text to all uppercase.
CTRL+M	Maximize the width of the script editor or restore it to its previous width.
CTRL+I	Opens the Function Wizard, which helps you add a function call at the insertion point in the script that is displayed in the InstallScript view.
ALT+K	Switches between adding and removing a bookmark in the line of script that currently contains the insertion point.
CTRL+K	Moves to the line in the script that contains the next bookmark.
CTRL+SHIFT+K	Moves to the line in the script that contains the previous bookmark.

Configuring Advanced Settings for InstallShield

One of the InstallShield program files is a file called **Settings.xml**. This file exposes some advanced machine-wide settings for InstallShield. When you install InstallShield, **Settings.xml** is installed to one of the following locations, depending on which language version of InstallShield you are using:

- **English**—*InstallShield Program Files Folder*\Support\0409
- **Japanese**—*InstallShield Program Files Folder*\Support\0411

The **Settings.xml** file for the Standalone build is installed in one of the following locations:

- **English**—*Standalone Build Program Files Folder*\Support\0409
- **Japanese**—*Standalone Build Program Files Folder*\Support\0411

In most cases, you should not modify the **Settings.xml** file. However, in some cases, you may need to make changes in this file. This section of the documentation describes some scenarios that would require **Settings.xml** changes:

- Changing the Timestamp Server for Digital Signatures
- Configuring the Compression Level for Files that Are Streamed into Setup.exe and ISSetup.dll
- Configuring the Maximum Size for .cab Files
- Modifying the List of Portable Executable Files for the Standalone Build
- Adding Support for XML Encoding Options
- Specifying the Location Where All Virtual Packages Should Be Built



Caution • *The* **Settings.xml** *file contains critical data; if this file is edited incorrectly, it can cause InstallShield to fail to work. Use extreme care when editing this file.*

Changing the Timestamp Server for Digital Signatures



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*

When you specify digital signature information for a release, InstallShield uses VeriSign's server (**http://timestamp.verisign.com/scripts/timstamp.dll**) as the default timestamp server during builds. InstallShield includes a machine-wide setting that lets you replace that default server with a different timestamp server. The setting also lets you disable timestamping.



Caution • *The following instructions require that you modify the* **Settings.xml** *file that is installed with InstallShield. This file contains critical data; if it is edited incorrectly, it can cause InstallShield to fail to work. Use extreme care when editing this file.*



Task ***To configure the timestamp server for digital signatures:***

1. Close InstallShield.
2. Find the **Settings.xml** file that is installed with InstallShield. **Settings.xml** is installed in one of the following locations, depending on which language version of InstallShield you are using:
   - **English**—*`InstallShield Program Files Folder`*\Support\0409
   - **Japanese**—*`InstallShield Program Files Folder`*\Support\0411

3. Create a back-up copy of the **Settings.xml** file, in case you later need to revert to the original version.
4. Use a text editor or XML file editor to open the **Settings.xml** file.
5. Search for the <DigitalSignature> element. It looks something like this:

   ```
   <DigitalSignature Timestamp="http://timestamp.verisign.com/scripts/timstamp.dll"/>
   ```

6. To override the timestamp server with a different one, set the value of the Timestamp attribute to the appropriate URL.

   To disable timestamping, use an empty value for the Timestamp attribute:

   ```
   <DigitalSignature Timestamp=""/>
   ```

   Note • *Disabling timestamping may affect how long your digital signature is considered to be valid.*

7. Save the **Settings.xml** file.
8. Ensure that your XML code is well formed; if it is not well formed, you may have problems using InstallShield. In most cases, you can identify improperly formed XML code by opening the **Settings.xml** file in Internet Explorer. You should be able to expand and contract the major elements of the file; if you cannot, check the code for errors.

Whenever you build a release that includes digital signature information, InstallShield sets the timestamp according to the setting that you configured.

Tip • *If you use the Standalone Build to build a release, update the* **Settings.xml** *file that is installed with the Standalone Build.* **Settings.xml** *is installed in one of the following locations, depending on which language version of InstallShield you are using:*

- ***English***—*Standalone Build Program Files Folder\Support\0409*
- ***Japanese***—*Standalone Build Program Files Folder\Support\0411*

Configuring the Compression Level for Files that Are Streamed into Setup.exe and ISSetup.dll

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

This information does not apply to custom compression, where only the files that are associated with one or more features are compressed into .cab files.

InstallShield includes a machine-wide setting that lets you specify the compression level that you want to use for files that are streamed into **Setup.exe** files and **ISSetup.dll** files at build time. Following are examples of files that may be streamed into the **Setup.exe** files and **ISSetup.dll** files:

- All of your product's files (if the release is one in which all of the files are compressed into the **Setup.exe** setup launcher)
- InstallShield prerequisite installations that have a location of Extract From Setup.exe
- The .NET Framework installation if it has a location of Extract From Setup.exe
- The Windows Installer installation if it has a location of Extract Engine From Setup.exe

InstallShield also lets you specify particular files (with or without wild-card characters) that should not be compressed when they are streamed into **Setup.exe** files or **ISSetup.dll** files.



Caution • *The following instructions require that you modify the* **Settings.xml** *file that is installed with InstallShield. This file contains critical data; if it is edited incorrectly, it can cause InstallShield to fail to work. Use extreme care when editing this file.*



Task ***To configure compression settings for streamed files:***

1. Close InstallShield.
2. Find the **Settings.xml** file that is installed with InstallShield. **Settings.xml** is installed in one of the following locations, depending on which language version of InstallShield you are using:
   - **English**—*`InstallShield Program Files Folder`*\Support\0409
   - **Japanese**—*`InstallShield Program Files Folder`*\Support\0411
3. Create a back-up copy of the **Settings.xml** file, in case you later need to revert to the original version.
4. Use a text editor or XML file editor to open the **Settings.xml** file.
5. Search for the <StreamCompression> element. It looks something like this:

   ```
   <StreamCompression exclude="*.CAB" compressionlevel="-1"/>
   ```

6. To prevent certain files or file types from being compressed when they are streamed into **Setup.exe** files or **ISSetup.dll** files, set the value of the exclude attribute to the names of those files. Note the following guidelines:
   - To specify more than one file, separate each file name with a comma.
   - To indicate a wild-card character, use an asterisk (*).

   For example, to specify that .cab files, .exe files, and a file called **test.txt** should not be compressed, set the value of the exclude attribute to as follows:

   ```
   <StreamCompression exclude="*.CAB,*EXE,test.txt" compressionlevel="-1"/>
   ```

   The default value is ***.CAB**, since .cab files are a compressed type of file.
7. Do one of the following:
   - If you want to use a compression level that offers a balance between the size of the compressed file and the time that is required to extract the compressed files at run time, set the value of the compressionlevel attribute to **-1**. This is the default value.

- To specify a particular compression level, set the value of the compressionlevel attribute to a number from 0 to 9, where 0 indicates no compression and 9 indicates maximum compression.

  In general, when you specify a number from 0 to 9, the higher the number that you specify, the smaller the resulting compressed file is, and the longer it may take to extract the files at run time.

8. Save the **Settings.xml** file.
9. Ensure that your XML code is well formed; if it is not well formed, you may have problems using InstallShield. In most cases, you can identify improperly formed XML code by opening the **Settings.xml** file in Internet Explorer. You should be able to expand and contract the major elements of the file; if you cannot, check the code for errors.

Whenever you build a compressed release for one of the applicable project types, InstallShield streams files into **Setup.exe** files and **ISSetup.dll** files according to the settings that you configured.

Tip • *If you use the Standalone Build to build a release, update the* **Settings.xml** *file that is installed with the Standalone Build.* **Settings.xml** *is installed in one of the following locations, depending on which language version of InstallShield you are using:*

- ***English***—`Standalone Build Program Files Folder\Support\0409`
- ***Japanese***—`Standalone Build Program Files Folder\Support\0411`

Configuring the Maximum Size for .cab Files

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

In addition, it is applicable only if you are building a compressed network image release in which all of the files are embedded in the single-file .msi package or the **Setup.exe** *setup launcher.*

This information does not apply to custom compression, where only the files that are associated with one or more features are compressed into .cab files.

The .cab file type has some limitations. For example, the maximum size of a single .cab file is 2 GB. In addition, some users have had trouble signing large .cab files and verifying the digital signature of large signed .cab files.

To work around these limitations, InstallShield enables you to specify on a machine-wide basis the maximum size for each .cab file that is built for a compressed network image release. When InstallShield is creating the .cab files for your release and it reaches the .cab file threshold that you configured, it splits the data into two or more .cab files, creating multi-part .cab files. Note that if you do not want InstallShield to create multi-part .cab files, you can configure InstallShield to store the data in a single .cab file.

Caution • *The following instructions require that you modify the* **Settings.xml** *file that is installed with InstallShield. This file contains critical data; if it is edited incorrectly, it can cause InstallShield to fail to work. Use extreme care when editing this file.*



Task ***To specify whether InstallShield should create multi-part .cab files and—if appropriate—specify the maximum .cab file size:***

1. Close InstallShield.
2. Find the **Settings.xml** file that is installed with InstallShield. **Settings.xml** is installed in one of the following locations, depending on which language version of InstallShield you are using:
   - **English**—*`InstallShield Program Files Folder`*\Support\0409
   - **Japanese**—*`InstallShield Program Files Folder`*\Support\0411
3. Create a back-up copy of the **Settings.xml** file, in case you later need to revert to the original version.
4. Use a text editor or XML file editor to open the **Settings.xml** file.
5. Search for the <CompressedNetworkCABSize> element. It looks something like this:

   ```
   <CompressedNetworkCABSize default="600"/>
   ```

6. Do one of the following:
   - To specify the maximum size for .cab files, type the size in MB as the value of the default attribute. In the aforementioned example, the maximum size is **600**. The value should be 2048 or less, since the maximum size of a .cab file is 2 GB (2048 MB).

     The default value is **600**.
   - If you do not want InstallShield to create multi-part .cab files, set the value of the default attribute to **-1**.
7. Save the **Settings.xml** file.
8. Ensure that your XML code is well formed; if it is not well formed, you may have problems using InstallShield. In most cases, you can identify improperly formed XML code by opening the **Settings.xml** file in Internet Explorer. You should be able to expand and contract the major elements of the file; if you cannot, check the code for errors.

Whenever you build a compressed network image release for one of the applicable project types, InstallShield creates the .cab files according to the requirement that you configured in the **Settings.xml** file. Depending on the value that you specified in the **Settings.xml** file, the **Media** table of the .msi package lists one or more .cab files.



Tip *• If you use the Standalone Build to build a release, update the* **Settings.xml** *file that is installed with the Standalone Build.* **Settings.xml** *is installed in one of the following locations, depending on which language version of InstallShield you are using:*

- ***English****—`Standalone Build Program Files Folder`\Support\0409*
- ***Japanese****—`Standalone Build Program Files Folder`\Support\0411*

Modifying the List of Portable Executable Files for the Standalone Build

The File Extensions tab on the Options dialog box in InstallShield lets you modify the file extensions that you would like InstallShield to consider to be portable executable (PE) files. InstallShield uses this list to determine how it should create components for dynamically linked files that adhere to the best practice component creation method.

If you are using the Standalone Build to build a release, the Options dialog box is not available. However, you can modify one of the files that is installed in a subfolder of the InstallShield Standalone Build Program Files folder if you want to modify the list of file extensions that the Standalone Build considers to be PE files. The following instructions explain how.



Caution • *The following instructions require that you modify the* **Settings.xml** *file that is installed with InstallShield. This file contains critical data; if it is edited incorrectly, it can cause InstallShield to fail to work. Use extreme care when editing this file.*



Task ***To modify the list of file extensions that the Standalone Build considers to be PE files:***

1. Close the Standalone Build.
2. Find the **Settings.xml** file that is installed with InstallShield. **Settings.xml** is installed in one of the following locations, depending on which language version of InstallShield you are using:
   - **English**—*Standalone Build Program Files Folder*\Support\0409
   - **Japanese**—*Standalone Build Program Files Folder*\Support\0411
3. Create a back-up copy of the **Settings.xml** file, in case you later need to revert to the original version.
4. Use a text editor or XML file editor to open the **Settings.xml** file.
5. Search for the PEFileExtensions element. It looks something like this:

   ```
   <PEFileExtensions default="EXE|DLL|OCX|VXD|CHM|HLB|TLB|AX">
   ```

6. Modify the list of file extensions for the default attribute as needed. Note that each file extension is separated with a vertical bar (|).
7. Save the **Settings.xml** file.
8. Ensure that your XML code is well formed; if it is not well formed, you may have problems using InstallShield. In most cases, you can identify improperly formed XML code by opening the **Settings.xml** file in Internet Explorer. You should be able to expand and contract the major elements of the file; if you cannot, check the code for errors.

The next time that you use the Standalone Build, it will use the updated list of file extensions.

Adding Support for XML Encoding Options

InstallShield lets you specify what type of encoding should be used for an XML file. This setting is available on the Advanced tab when you select the XML file in the XML File Changes view. If the type of encoding that you want to use is not included in the list of available encoding options, you can modify one of the files that is installed in a subfolder of the InstallShield Program Files folder to add additional options. You can add any encoding that is supported by MSXML. The following instructions explain how.



Caution • *The following instructions require that you modify the* **Settings.xml** *file that is installed with InstallShield. This file contains critical data; if it is edited incorrectly, it can cause InstallShield to fail to work. Use extreme care when editing this file.*



Task ***To add additional encoding options to the Encoding list on the Advanced tab in the XML File Changes view:***

1. Close InstallShield.
2. Find the **Settings.xml** file that is installed with InstallShield. **Settings.xml** is installed in one of the following locations, depending on which language version of InstallShield you are using:
   - **English**—*`InstallShield Program Files Folder`*\System\0409
   - **Japanese**—*`InstallShield Program Files Folder`*\System\0411
3. Create a back-up copy of the **Settings.xml** file, in case you later need to revert to the original version.
4. Use a text editor or XML file editor to open the **Settings.xml** file.
5. Search for the ISXML element and its child elements. They look something like this:

```
<ISXML>
  <Encodings>
    <Encoding>WINDOWS-1250</Encoding>
    <Encoding>WINDOWS-1251</Encoding>
    <Encoding>WINDOWS-1252</Encoding>
    <Encoding>WINDOWS-1253</Encoding>
    <Encoding>WINDOWS-1254</Encoding>
    <Encoding>WINDOWS-1255</Encoding>
    <Encoding>WINDOWS-1256</Encoding>
    <Encoding>WINDOWS-1257</Encoding>
    <Encoding>WINDOWS-1258</Encoding>
    <Encoding>ISO-8859-1</Encoding>
    <Encoding>ISO-8859-2</Encoding>
    <Encoding>ISO-8859-3</Encoding>
    <Encoding>ISO-8859-4</Encoding>
    <Encoding>ISO-8859-5</Encoding>
    <Encoding>ISO-8859-6</Encoding>
    <Encoding>ISO-8859-7</Encoding>
    <Encoding>ISO-8859-8</Encoding>
    <Encoding>ISO-8859-9</Encoding>
    <Encoding>US-ASCII</Encoding>
    <Encoding>UNICODE-1-1-UTF-8</Encoding>
    <Encoding>UNICODE-2-0-UTF-16</Encoding>
    <Encoding>UNICODE-2-0-UTF-8</Encoding>
    <Encoding>ISO-10646-UCS-2</Encoding>
```

```
    <Encoding>UCS-4</Encoding>
    <Encoding>UCS-2</Encoding>
    <Encoding>UTF-16</Encoding>
    <Encoding Default="true">UTF-8</Encoding>
  </Encodings>
</ISXML>
```

6. Between the opening and closing Encodings tags, add a new line such as this:

   <Encoding>Type_of_Encoding</Encoding>

   where *Type_of_Encoding* indicates the encoding that you want to be available. This should be the value that InstallShield should use for the encoding attribute of your XML document.

7. Save the **Settings.xml** file.

8. Ensure that your XML code is well formed; if it is not well formed, you may have problems using InstallShield. In most cases, you can identify improperly formed XML code by opening the **Settings.xml** file in Internet Explorer. You should be able to expand and contract the <ISXML> and <Encodings> elements; if you cannot, check the code for errors.

The next time that you open the XML File Changes view in InstallShield, you will see the encoding type that you added as one of the available options in the **Encoding used for a new file** list on the Advanced tab for an XML file in the XML File Changes view.

Specifying the Location Where All Virtual Packages Should Be Built

Edition • *Support for virtualization is included in the Premier Edition of InstallShield.*

When InstallShield converts an .msi package to a virtual package, it saves the virtual package in a subfolder of the folder that contains the .msi file by default. In some cases, you may want to specify a different location for generating all of the virtual packages. For example, if the .msi packages that you are converting are in a read-only location, you may need to override the default virtual package location with an existing writable location.

InstallShield includes a machine-wide setting that lets you specify an existing writable location where all virtual packages should be built. The following instructions explain how to configure this global setting.

Caution • *The following instructions require that you modify the* **Settings.xml** *file that is installed with InstallShield. This file contains critical data; if it is edited incorrectly, it can cause InstallShield to fail to work. Use extreme care when editing this file.*

Task ***To specify the location where all virtual packages should be built:***

1. Close InstallShield.

2. Find the **Settings.xml** file that is installed with InstallShield. **Settings.xml** is installed in one of the following locations, depending on which language version of InstallShield you are using:

   - **English**—*InstallShield Program Files Folder*\System\0409

- **Japanese**—*InstallShield Program Files Folder*\System\0411

3. Create a back-up copy of the **Settings.xml** file, in case you later need to revert to the original version.
4. Use a text editor or XML file editor to open the **Settings.xml** file.
5. Search for the <Virtualization> element and its child element. They look something like this:

```
<Virtualization>
  <!-- Instructions on how to specify a global path -->
  <GlobalBuildRedirectFolder></GlobalBuildRedirectFolder>
</Virtualization>
```

6. Enter the global path as the text content for the <GlobalBuildRedirectFolder> element. For example, to create the virtual packages in a subfolder of **\\NetworkFolder\VirtualPackages**, use the following:

   <GlobalBuildRedirectFolder>\\NetworkFolder\VirtualPackages</GlobalBuildRedirectFolder>

   Note that the path must already exist. Do not enclose the path within quotation marks.
7. Save the **Settings.xml** file.
8. Ensure that your XML code is well formed; if it is not well formed, you may have problems using InstallShield. In most cases, you can identify improperly formed XML code by opening the **Settings.xml** file in Internet Explorer. You should be able to expand and contract the <Virtualization> element; if you cannot, check the code for errors.

Whenever you convert an .msi package to a virtual package in InstallShield, InstallShield uses the path that you specified.

Sharing Virtual Machine Settings for Distribution of Releases

Edition • *The ability to distribute releases to virtual machines is available in the Premier edition of InstallShield.*

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

You can configure releases in your projects so that after each successful build of your installation, InstallShield automatically reverts a virtual machine (VM) to a designated snapshot, powers on the VM, and copies your installation to the VM to make it available for testing. You can also alternately perform these testing preparation steps on demand at any time.

When you use the Configuration setting on the Events tab for a selected release in the Releases view to configure VM details, InstallShield writes the data that you specify to a file called **VMConfigurations.xml**. This **VMConfigurations.xml** file is a machine-wide file that you can configure once and then use it in other InstallShield projects, as well as share it with other team members. You can also use this file with the Standalone Build.

The following instructions explain how to share this global file with others.



Task ***To share the customized VM configuration settings file with another machine that has InstallShield or the Standalone Build:***

1. On your development machine, configure the VM settings for a release in one of your projects. To learn how, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.
2. Close InstallShield.
3. Find the **VMConfigurations.xml** file that is installed with InstallShield, and make a copy of it for the other machine. **VMConfigurations.xml** is installed in one of the following locations, depending on which language version of InstallShield you are using:
   - **English**—*InstallShield Program Files Folder*\Support\0409
   - **Japanese**—*InstallShield Program Files Folder*\Support\0411
4. Create a back-up copy of the **VMConfigurations.xml** file that is on the other machine, in case you later need to revert to the original version, and then overwrite the installed version with the one from your development machine.

   If you use the Standalone Build to build a release, update the **VMConfigurations.xml** file that is installed with the Standalone Build. **VMConfigurations.xml** is installed in one of the following locations, depending on which language version of InstallShield you are using:
   - **English**—*Standalone Build Program Files Folder*\Support\0409
   - **Japanese**—*Standalone Build Program Files Folder*\Support\0411



Tip *• If you make any manual changes to the XML file in a text editor, ensure that your XML code is well formed; if it is not well formed, you may have problems using InstallShield. In most cases, you can identify improperly formed XML code by opening the* **VMConfigurations.xml** *file in Internet Explorer. You should be able to expand and contract the <VMConfigurations> element; if you cannot, check the code for errors.*

Upgrading from Earlier InstallShield Versions

When you start using InstallShield 2016, you can upgrade installation projects that you created using earlier versions or different InstallShield products. This section discusses those upgrades.

- Upgrading Projects from InstallShield 2014 or Earlier
- Upgrading Projects from InstallShield 2013 or Earlier
- Upgrading Projects from InstallShield 2012 Spring or Earlier
- Upgrading Projects from InstallShield 2012 or Earlier
- Upgrading Projects from InstallShield 2011 or Earlier
- Upgrading Projects from InstallShield 2010 or Earlier
- Upgrading Projects from InstallShield 2009 or Earlier

- Upgrading Projects from InstallShield 2008 or Earlier
- Upgrading Projects from InstallShield 12 or Earlier
- Upgrading Projects from InstallShield 11.5 or Earlier

Upgrading Projects from InstallShield 2014 or Earlier

The following information describes possible upgrade issues that may occur when you upgrade projects that were created with InstallShield 2014 and earlier to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from InstallShield 2014 or earlier to InstallShield 2016.

General Information about Upgrading Projects that Were Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open an project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .775 (for an .ism project) or .2014 (for an .issuite project) before converting it. Delete the .775 or .2014 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2016: InstallShield 2014 and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2016.

Changes to the List of Supported Versions of Windows for Target Systems

Windows XP SP3 and Windows Server 2003 SP2 are now the minimum versions of Windows that are required for target systems that run the installations that are created in InstallShield; this applies to all project types.

Removal of Support for Digitally Signing with .spc and .pvk Files

InstallShield no longer has support for using .spc and .pvk files to digitally sign files at build time.

If you configured a release or patch in InstallShield 2014 or earlier to be digitally signed at run time with .spc and .pvk files, and then you try to open that project in InstallShield 2016, InstallShield displays upgrade warning -6048 (for a release), -6049 (for a patch), or -6050 (for a QuickPatch project). The warning explains that InstallShield is removing the .pvk file and the associated password from your project during the upgrade.

Before you can successfully build the release or the patch in InstallShield 2016, you will need to remove the .spc reference from the release or patch configuration. You can replace it with a .pfx certificate, or with a reference to a certificate in a certificate store. To learn more, see:

- Digital Signing and Security
- Digitally Signing a Release and Its Files at Build Time
- Signing a Patch Package
- Signing a QuickPatch Package

If you try to build a release or a patch without removing the .spc reference from it, InstallShield displays build error -7347, stating that the .spc file must be removed.

To learn how to convert an .spc file and a .pvk file to a .pfx file, see Digital Signing and Security.

Automation Interface Changes for Digital Signature Support

The ISWiRelease object no longer includes support for the following read-write properties:

- CorrespondingPrivateKey
- SoftwarePublishingCredentials

If you call those properties, you will encounter an error. Those properties have been replaced with the read-write string property DigitalCertificateInfo.

To learn more, see ISWiRelease Object.

Removal of SignTool.exe and Signcode.exe from InstallShield Installation

SignTool.exe and **Signcode.exe** are no longer installed on your machine when you install InstallShield. If you want to digitally sign your files manually, consider using **SignTool.exe**, which is installed with Visual Studio and included in the Microsoft Windows Software Development Kit (SDK).

Changes to the PowerShell Support in Basic MSI and InstallScript MSI Installations

The PowerShell custom action support in Basic MSI and InstallScript MSI installations has been revised. The support no longer uses the Windows Installer property **IS_PS_EXECUTIONPOLICY** to indicate the name of the PowerShell execution policy that you want to be used to run PowerShell custom actions on target systems. Setting this property at run time no longer has any effect on the installation.

Changes to Default Eligibility Conditions for .msi Packages in Advanced UI and Suite/Advanced UI Projects

To make it possible to support shared packages in Advanced UI and Suite/Advanced UI projects, InstallShield no longer needs the previously required MSI Package eligibility conditions for .msi packages. Now you can limit the use of eligibility conditions of .msi packages to check for run-time environment requirements such as platform.

Now when you add an .msi package to an Advanced UI or Suite/Advanced UI project in InstallShield 2016, InstallShield no longer creates an MSI Package eligibility condition for that package by default. This behavior applies to new InstallShield 2016 projects, as well as projects that you created in InstallShield 2014 or earlier and then upgrade to InstallShield 2016.

In addition, if you upgrade an InstallShield 2014 or earlier project that contains the old default MSI Package eligibility condition for an .msi package to InstallShield 2016, InstallShield removes the unnecessary part of the condition from the Eligibility Condition setting for the package.

In some cases, if you customized the default MSI Package eligibility condition in InstallShield 2014 or earlier, InstallShield 2016 may remove it during the upgrade; in other cases, InstallShield leaves it as is. The actual upgrade behavior depends on the customization. For example, if you simply added a platform condition, InstallShield may be able to remove the original MSI Package part of the condition, leaving just the platform condition. However, if the customization is too complex, InstallShield leaves the condition as is.

It is recommended that you review the eligibility conditions for your .msi packages after upgrading to InstallShield 2016 to ensure that they are configured as expected.

If you have customized or somehow otherwise edited the default MSI Package eligibility condition, InstallShield leaves your custom condition during the upgrade. It is recommended that you review the eligibility conditions for your .msi packages to ensure that they are configured as expected.

Previously, the default MSI Package condition for the Eligibility Condition setting of .msi packages used asterisks (*) in the condition's Product Code and Product Version settings as placeholders for the package's own product code and product version.

Changes to the Interpretation of Ampersands in the Wizard Interface of Advanced UI and Suite/Advanced UI Installations

The interpretation of ampersands in certain areas of the wizard interface of Advanced UI and Suite/Advanced UI projects has been modified.

For new InstallShield 2016 projects, as well as projects that you created in InstallShield 2014 or earlier and then upgrade to InstallShield 2016, if you use an ampersand (&) in any of the following areas of the wizard interface of your installation, the installation now displays the ampersand as a literal character instead of interpreting it as a prefix character for a keyboard shortcut. The change also applies if any of these strings include a property that resolves to a value that contains an ampersand.

- The string in the header area of a wizard page or secondary window—This is configured in the Title setting for a wizard page or window in the Wizard Interface view.
- The caption bar of wizard pages—This is configured in the Wizard Caption setting for the Wizard Pages node in the Wizard Interface view.
- The supplemental explanation area of a command link control—This is configured in the Note setting for a wizard page or window.
- The alternate text for an image control—This is configured in the Alt Text setting of an image control on a wizard page or window.

Furthermore, for most label controls, True is now selected by default for the SS_NOPREFIX subsetting under the Style setting; previously False was selected by default. A True value for this subsetting ensures that any ampersands that are included in the string entries for such controls are not inadvertently interpreted as keyboard shortcuts, and that ampersands in the control's strings are displayed as ampersands in the wizard interface.

These SS_NOPREFIX changes apply to the built-in default and predefined wizard pages and windows that are available for new InstallShield 2016 projects. If you upgrade an InstallShield 2014 or earlier project to InstallShield 2016, the SS_NOPREFIX settings are changed automatically for the standard built-in default wizard page and windows. However, no changes are made automatically to custom wizard pages and windows—including the predefined wizard pages—when you upgrade these projects to InstallShield 2016; if you want to change the ampersand interpretation of these, you can do so manually

Previously, to work around problems with an ampersand being used inadvertently to designate a keyboard shortcut in any of the aforementioned areas of the wizard interface, and instead show a single ampersand, it was possible to use an escaped ampersand (&&) in place of a single ampersand. However, if you upgrade a project that contains one or more instances of those workarounds to InstallShield 2016, the escaped ampersand is now displayed literally as two ampersands. Thus, it may be necessary to review the use of ampersands in the wizard interface of your projects and make changes as needed.

For more information, see Designating Keyboard Shortcuts and Using Ampersands (&) in the Wizard Interface.

Trialware Support

InstallShield no longer includes support for creating the Try and Die or the Try and Buy/Product Activation types of trialware. The Trialware view is no longer included in InstallShield.

Upgrading Projects from InstallShield 2013 or Earlier

The following information describes possible upgrade issues that may occur when you upgrade projects that were created with InstallShield 2013 and earlier to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from InstallShield 2013 or earlier to InstallShield 2016.

General Information about Upgrading Projects that Were Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open an project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .774 (for an .ism project) or .2013 (for an .issuite project) before converting it. Delete the .774 or .2013 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2016: InstallShield 2013 and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2016.

Upgrading Projects from InstallShield 2012 Spring or Earlier

The following information describes possible upgrade issues that may occur when you upgrade projects that were created with InstallShield 2012 Spring and earlier to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from InstallShield 2012 Spring or earlier to InstallShield 2016.

General Information about Upgrading Projects that Were Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open an project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .773 (for an .ism project) or .2012.4 (for an .issuite project) before converting it. Delete the .773 or .2012.4 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2016: InstallShield 2012 Spring and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2016.

Changes that Affect All Projects (New and Upgraded Projects)

This section describes changes that affect both new projects and projects that are upgraded from earlier versions of InstallShield.

Change in Requirements for Target Systems

InstallShield no longer supports the creation of installations for Windows 2000 systems. If end users have Windows 2000 on their machine and they try to run an installation that was built with InstallShield 2013, the installation may run successfully. Or, unexpected results may occur, unless the project includes launch conditions that prevent end users from running the installation on this legacy operating system.

For Basic MSI and InstallScript MSI projects, you may want to consider adding a launch condition that displays a message if end users are running your installation on a Windows 2000 system. For InstallScript projects, you may want to consider adding InstallScript code that uses the SYSINFO structure variable to check for Windows 2000 systems, and then display a message if appropriate.

Advanced UI and Suite/Advanced UI installations now require at least Windows Vista and Windows Server 2008.

InstallShield no longer supports the creation of installations for mobile devices. Thus, the Mobile Devices view, the Smart Device project type, the Palm OS Object, and the Windows Mobile Object are no longer included in InstallShield. If you try to upgrade a Smart Device project from InstallShield 2012 Spring or earlier to InstallShield 2016, InstallShield 2016 displays an error message and fails to open the project. If you upgrade a project from InstallShield 2012 Spring or earlier to InstallShield 2016, and if the project targets desktop platforms and contains other mobile device support, InstallShield removes the mobile device support during the upgrade and logs a warning.

Change in Requirements for Systems Running InstallShield

The minimum operating system requirement for running InstallShield, the Standalone Build, and the InstallShield Developer Installation Manifest (DIM) Editor is now Windows XP SP3 or Windows Server 2003 SP2. Previously, the minimum operating system requirement was an RTM version of either of these operating systems.

Changes for the Virtualization Pack

The Virtualization Pack is included as part of the standalone version of the Premier Edition of InstallShield. It is no longer available as an add-on for the Professional edition of InstallShield.

Repackager Project Conversion Tool No Longer Included

The Repackager is no longer included in the Premier edition of InstallShield. Repackager is now available only with AdminStudio.

Changes to the XML Schema for Advanced UI and Suite/Advanced UI Project Files (.issuite)

Advanced UI and Suite/Advanced UI project files (.issuite) are XML-based files. The underlying XML schema for the user interface part of the file has changed significantly in InstallShield 2016. Many attributes have been moved to child elements. Conditions, actions, and validations are fundamentally different: element collections are used instead of attribute strings. The new schema is more explicit about what is configured for the user interface of the installation.

If you create a new Advanced UI or Suite/Advanced UI project in InstallShield 2016, InstallShield automatically uses the new XML schema for the .issuite file. If you upgrade a project from InstallShield 2012 Spring or earlier to InstallShield 2016, InstallShield automatically updates the XML schema.

New Predefined Path Variables for Built-in DLL Custom Actions

InstallShield includes two new predefined path variables in projects: ISRedistPlatformDependentFolder and ISRedistPlatformDependentExpressFolder. The default values for these folders refer to subfolders in the *InstallShield Program Files Folder*\Redist folder, which contains 32-bit versions of built-in InstallShield custom action DLLs. If you create a new InstallShield 2016 project or upgrade an InstallShield 2012 Spring or later project to InstallShield 2016, InstallShield automatically includes these two predefined path variables in your project. In addition, InstallShield also uses these path variables in paths such as the source location of built-in InstallShield DLL custom actions; this applies to new projects as well as upgraded projects. Previously, InstallShield used either of the following locations as the folder that contained the DLL files: <ISProductFolder>\Redist\Language Independent\i386 or <ISProductFolder>\Redist\Language Independent\i386 Express. This change is noted for informational purposes.

Automation Interface Changes

If you use the automation interface with InstallShield or the Standalone Build, update your existing code to reflect the new ProgID: IswiAutoAutomation Interface Version.ISWiProject. The Standalone Automation Interface uses the same ISWiAutomationAutomation Interface Version.dll file that InstallShield uses, but it is installed to a different location.

Note that if you install the Standalone Build on the same machine as InstallShield, the last ISWiAutomationAutomation Interface Version.dll file that is registered is the one that is used.

Changes that Affect New Projects but Not Upgraded Projects

This section describes changes to InstallShield that may affect new projects but not projects that are upgraded from earlier versions. Note that you may need to make manual changes to upgraded projects.

Changes to the Default Wizard Format for the Suite/Advanced UI and Advanced UI Wizard Interface

If you create a new Suite/Advanced UI or Advanced UI project in InstallShield 2016, the default value for the Wizard Format setting is Glass, a new option. If you upgrade a Suite/Advanced UI or Advanced UI project from an earlier version of InstallShield to InstallShield 2016, InstallShield does not change the value of the Wizard Format setting; it leaves whatever value was selected in the earlier version of InstallShield.

You can configure your project to use to a different wizard format if appropriate. To do so, change the value of the Wizard Format setting that is displayed in the Wizard Interface view when the Wizard Pages node is selected.

For more information, see Selecting the Format for the Wizard Interface.

Changes to the Font Color of Navigation Button Text in Advanced UI and Suite/Advanced UI Installations

In some scenarios, if you upgrade an Advanced UI or Suite/Advanced UI project from InstallShield 2012 Spring or earlier to InstallShield 2016, the text on the navigation buttons may no longer be legible: the colors for the text and the buttons may not have enough contrast. This may occur if light font color is specified for the text style that is used for navigation button text, and if the target system's colors are configured to use a light color for buttons on windows.

Beginning with InstallShield 2016, Advanced UI and Suite/Advanced UI installations no longer ignore the font color that is specified for the text style of text that is used on navigation buttons. Thus, when you upgrade your Advanced UI or Suite/Advanced UI project to InstallShield 2016, you may need to adjust the color of the text style that is used

for the navigation button text. To do so, in the Wizard Interface view, click the Wizard Pages node, and note the text style that is selected for the Navigation Text Style setting. Then, find that text style under the Styles node in this view, and adjust its settings as needed.

In InstallShield 2012 Spring or earlier, Advanced UI and Suite/Advanced UI projects ignored the text style color for navigation button text. At run time, the installation used the font color that Windows used for navigation buttons; the color was typically black.

New Support for Shell Properties of Shortcuts

The Shortcuts view in InstallShield now has built-in support for setting several Windows Shell properties of shortcuts:

- The Pin to Windows 8 Start Screen setting sets the System.AppUserModel.StartPinOption property by using the following GUID and property ID combination:

  9F4C2855-9F79-4B39-A8D0-E1D42DE1D5F3, 12

- The Prevent Pinning option in the Shell Properties setting sets the System.AppUserModel.PreventPinning property by using the following GUID and property ID combination:

  9F4C2855-9F79-4B39-A8D0-E1D42DE1D5F3, 9

- The Do Not Highlight as New option in the Shell Properties setting sets the System.AppUserModel.ExcludeFromShowInNewInstall property by using the following GUID and property ID combination:

  9F4C2855-9F79-4B39-A8D0-E1D42DE1D5F3, 8

Windows Installer may generate errors if one or more of these property names (instead of the GUID and property ID) are used in a package that is run on a version of Windows that does not have support for these properties.

If you used the Shell Properties setting in InstallShield 2012 Spring or earlier to configure Shell properties for a shortcut, and you upgrade the project to InstallShield 2016, InstallShield does not automatically replace the property names with the appropriate GUID and property ID combinations. To quickly switch to the GUID and property ID combination, consider deleting the old configuration in the Shell Properties view after upgrading your project, and then using the new support to reconfigure one or more of these properties. As an alternative, you can manually override the entry with the appropriate GUID and property ID. To learn more about the built-in support in InstallShield, see Setting Shell Properties for a Shortcut.

This support is available for the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

Upgrading Projects from InstallShield 2012 or Earlier

The following information describes possible upgrade issues that may occur when you upgrade projects that were created with InstallShield 2012 and earlier to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from InstallShield 2012 or earlier to InstallShield 2016.

General Information about Upgrading Projects that Were Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open an project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .772 (for an .ism project) or .2012 (for an .issuite project) before converting it. Delete the .772 or .2012 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2016: InstallShield 2012 and earlier, InstallShield 12 and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2016.

Changes that Affect All Projects (New and Upgraded Projects)

This section describes changes that affect both new projects and projects that are upgraded from earlier versions of InstallShield.

New Default Behavior When Launching an Earlier Version or the Same Version of a Suite/Advanced UI Installation on Target Systems

InstallShield now includes two Suite Installed conditions in each Advanced UI and Suite/Advanced UI project by default:

- A new Suite Installed exit condition prevents end users from being able to install the current version of the Advanced UI or Suite/Advanced UI installation over a future newer version of the same Advanced UI or Suite/Advanced UI installation.
- A new Suite Installed mode condition now causes the Advanced UI or Suite/Advanced UI installation to run in first-time installation mode if end users install a new version of the Advanced UI or Suite/Advanced UI installation over an older version of the same Advanced UI or Suite/Advanced UI installation.

These new default conditions are available in all new Suite/Advanced UI projects. If you upgrade an InstallShield 2012 Suite project to InstallShield 2016, InstallShield automatically adds these default conditions to the project. To learn more about these conditions, see Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed.

Previously, if an end user installed a particular version of a Suite installation on a target system on which a newer version of the Suite was already installed, the installation could permit the end user to install the older Suite version over the newer version. In addition, if an end user installed a particular version of a Suite installation on a target system on which the same version of the Suite was already installed, the Suite installation could run in first-time installation mode. Furthermore, if an end user installed a new version of a Suite installation on a target system on which an older version of the Suite was already installed, the Suite installation could run in maintenance mode.

Automation Interface Changes

If you use the automation interface with InstallShield or the Standalone Build, update your existing code to reflect the new ProgID: IswiAutoAutomation Interface Version.ISWiProject. The Standalone Automation Interface uses the same ISWiAutomationAutomation Interface Version.dll file that InstallShield uses, but it is installed to a different location.

Note that if you install the Standalone Build on the same machine as InstallShield, the last ISWiAutomationAutomation Interface Version.dll file that is registered is the one that is used.

Changes that Affect New Projects but Not Upgraded Projects

This section describes changes to InstallShield that may affect new projects but not projects that are upgraded from earlier versions. Note that you may need to make manual changes to upgraded projects.

Changes to the Default Behavior of New Combo Boxes in the Wizard Interface of Suite/Advanced UI Projects

If you create a new combo box control on a wizard page or a secondary window in a Suite/Advanced UI project in InstallShield 2016, the control is a box that contains a drop-down list of predefined values. The box is also a text box that lets end users enter a custom value. Previously, if you added a new combo box control in InstallShield 2012, the control contained a drop-down list, but it was not also a text box; that is, end users could not enter a custom value.

If you upgrade the project from InstallShield 2012 to InstallShield 2016, and if the Suite/Advanced UI wizard interface includes a combo box, the combo box is left as a drop-down list of predefined values, but it is not also a text box. To change the control to a drop-down list with the text box, set the CBS_DROPDOWNLIST style for this control to False.

Upgrading Projects from InstallShield 2011 or Earlier

The following information describes possible upgrade issues that may occur when you upgrade projects that were created with InstallShield 2011 and earlier to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from InstallShield 2011 or earlier to InstallShield 2016.

General Information about Upgrading Projects that Were Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open a project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .771 before converting it. Delete the .771 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2016: InstallShield 2011 and earlier, InstallShield 12 and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2016.

Changes that Affect All Projects (New and Upgraded Projects)

This section describes changes that affect both new projects and projects that are upgraded from earlier versions of InstallShield.

Changes in Behavior for Some MSI APIs That Are Called in InstallScript Custom Actions

A change was made to how the InstallScript engine calls Windows Installer API such as **MsiGetProperty**. The engine now calls the Windows Installer APIs directly instead of using wrapper APIs that in turn call the Windows Installer APIs. This change was made to resolve issues with buffer handling and buffer sizes. It applies to all new projects that you create in InstallShield 2016, as well as projects that you have upgraded from earlier versions of InstallShield to InstallShield 2016.

As a result of this change, you can use the MSI install handle that is passed to a custom action function with all Windows Installer APIs that require an install handle. Previously, the handle was managed by the InstallScript engine, and it could not be passed directly to Windows Installer APIs.

Also as a result of this change, any Windows Installer APIs that require a buffer size to be specified now fail if the buffer size is not correctly specified. Note that 0 is not a valid size; thus, ensure that your code does not pass a zero value for the size of the buffer.

If you have existing InstallScript custom actions that call Windows Installer APIs, ensure that a large enough buffer size is specified; if it is not, unexpected results could occur.

The following sample code demonstrates how to retrieve the value of a Windows Installer property value string in InstallScript and increase the size of the buffer if necessary.

```
prototype STRING MyGetProperty (HWND, STRING);
////////////////////////////////////////////////////////////////////////////
//   MyGetProperty
//
//   Return an MSI property string value
//   Input Parameters:
//        hMSIHandle:      Handle to the currently running MSI database
//        szPropertyName:  Name of property to retrieve
//    Output:
//        String containing the property value
////////////////////////////////////////////////////////////////////////////

function STRING MyGetProperty (hMSIHandle, szPropertyName)
    NUMBER  nvBuf, nResult;
    STRING  szReturn;
begin

//Retrieve the size of the property value
    szReturn = "";

    nResult = MsiGetProperty (hMSIHandle, szPropertyName, szReturn, nvBuf);

    if (ERROR_MORE_DATA = nResult) then
        nvBuf = nvBuf + 1;   //increment buffer size for terminating null

        //Retrieve the property value
        nResult = MsiGetProperty (hMSIHandle, szPropertyName, szReturn, nvBuf);

        if (nResult != ERROR_SUCCESS) then
            szReturn = "";
        endif;
    endif;
    return szReturn;
end;
```

Note that if the script had used **if (ERROR_SUCCESS == nResult) then** instead of **if (ERROR_MORE_DATA == nResult) then**, unexpected results could occur.

Build Warning -7235 for Basic MSI Projects

By default, software identification tagging is enabled in all Basic MSI projects. This applies to new projects that you create in InstallShield 2016, as well as projects that you have upgraded from earlier versions of InstallShield to InstallShield 2016.

If you build a release in a Basic MSI project without entering data in the required identification tag settings (the Unique ID, Tag Creator, and Tag Creator ID settings in the General Information view), and you leave tagging enabled in the project, build warning -7235 occurs. This build warning explains that the software identification tag could not be created and included in the installation because a specific required setting was left blank. To resolve this warning, enter appropriate value in each specific setting, or select No for the Use Software Identification Tag setting in the General Information view.

COM Extraction Changes

InstallShield supports a new monitoring method for COM extraction. If you are using InstallShield on a Windows Vista or later system or a Windows Server 2008 or later system, this new method is used by default. The method uses a kernel driver to monitor the areas of the registry that are modified during dynamic COM extraction at build time and static COM extraction at design time. It combines the advantages that the earlier methods provided, allowing the DLL to read existing registries entries and preventing changes to the build machine.

If necessary, you can switch between the three different COM extraction methods by setting the value data of the UseAPIRegistryHooks registry value, which is in the registry key HKEY_LOCAL_MACHINE\SOFTWARE\InstallShield\RegSpy (on 32-bit machines) or HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\InstallShield\RegSpy (on 64-bit machines). Possible REG_DWORD value data are:

- **0**—Use API hooking to read existing registry entries for the DLL.
- **1**—Use registry redirection to prevent making changes to the registered DLLs on the build machine. If the value is not set, this is the default behavior on Windows XP and Windows Server 2003 systems.
- **2**—Use the new kernel mode monitoring, which combines the advantages of both of the other methods. If the value is not set, this is the default behavior on Windows Vista and later systems and on Windows Server 2008 and later systems.

This functionality applies to the following project types: Basic MSI, DIM, InstallScript MSI, and Merge Module.

String Entries in Merge Module Projects

Each string identifier that is created in a Merge Module project now contains the Merge Module's module ID GUID. This applies to all new string identifiers that are created in all new Merge Module projects, as well as new string identifiers that are created in existing Merge Module projects that were upgraded from InstallShield 2011 or earlier to InstallShield 2016. Note that if you upgrade a Merge Module project from InstallShield 2011 or earlier to InstallShield 2016, the module ID GUID is not added to the string identifiers of any existing string entries.

The use of the GUID in string identifiers of Merge Module project string entries helps to minimize or eliminate conflicts that would occur if the same string identifier is used in a Merge Module project and also in an installation project that contains that Merge Module, and if different values are assigned to those two string identifiers.

Upgrading Projects from InstallShield 2010 or Earlier

The following information describes possible upgrade issues that may occur when you upgrade projects that were created with InstallShield 2010 and earlier to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from InstallShield 2010 or earlier to InstallShield 2016.

General Information about Upgrading Projects that Were Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open a project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .770 before converting it. Delete the .770 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2016: InstallShield 2010 and earlier, InstallShield 12 and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2016.

Changes that Affect All Projects (New and Upgraded Projects)

This section describes changes that affect both new projects and projects that are upgraded from earlier versions of InstallShield.

End of Integration Support for Visual Studio 2003 and Earlier

If you want to create, edit, and build your InstallShield projects directly within Visual Studio, you must use Visual Studio 2005 or later. InstallShield can no longer be integrated with Visual Studio 2003 or earlier.

Deprecation of InstallScript Objects

InstallScript objects have been deprecated in favor of InstallShield prerequisites. In a future release, InstallShield will no longer be able to create or consume InstallScript objects, and no predefined InstallScript objects will be provided. Furthermore, the Merge Module Holder Object will not be available. The recommended alternative for InstallScript objects is InstallShield prerequisites. You can use the InstallShield Prerequisite Editor to begin creating your own InstallShield prerequisites so that you are ready to use them once the InstallScript object technology becomes obsolete. You can share these InstallShield prerequisites among InstallScript, InstallScript MSI, and Basic MSI projects.

InstallShield Builds Only Unicode Versions of Setup.exe and Update.exe; Ability to Create ANSI Versions Is No Longer Available

Now all **Setup.exe** and **Update.exe** files that are built in all project types in InstallShield are Unicode. This applies to all new Basic MSI, InstallScript, InstallScript MSI, and QuickPatch projects that you create in InstallShield 2016. It also applies to all projects that you have upgraded from earlier versions of InstallShield to InstallShield 2016. Therefore, the settings that previously enabled you to specify whether you wanted to build a Unicode version or an ANSI version of the setup launcher have been removed:

- The Setup Launcher Type setting on the Setup.exe tab for a release in the Releases view has been removed from Basic MSI projects.

- The Update Launcher Type setting was removed from Basic MSI, InstallScript MSI, and QuickPatch projects.

  In Basic MSI and InstallScript MSI projects, this setting was on the Advanced tab for a patch configuration in the Patch Design view.

  In QuickPatch projects, this setting was on the Advanced tab in the Build Settings area of the General Information view.

Changes to Win32 API Definitions

Now that the InstallScript engine has been updated to support Unicode, the Win32 API functions that are prototyped in the InstallScript header file **ISRTWindows.h** have been updated. Where applicable, wide (W) versions of API prototypes have been added in addition to existing ANSI (A) definitions. For some prototypes, no A or W version is specified; in these cases, the engine now attempt to use the W version. Previously, the A version was used.

If you upgrade an InstallShield 2010 or earlier project to InstallShield 2016, it is possible that these new prototypes could conflict with user-defined prototypes of the same APIs. It is recommended that the InstallScript-provided prototypes be used if possible. However, if you want to use your own prototypes for these Windows APIs instead of the ones that are now prototyped for InstallScript, add ISINCLUDE_NO_WINAPI_H to the list of preprocessor definitions: On the Build menu, click Settings. On the Compile/Link tab, in the Preprocessor Defines box, enter ISINCLUDE_NO_WINAPI_H. Otherwise, you may encounter compile errors.

Ensuring that Your InstallScript Code Supports Unicode

If you use any user-defined Win32 APIs or other external DLL prototypes in your InstallScript code, and the APIs have versions that support Unicode string input, update the prototypes to use BYVAL/BYREF WSTRING or WPOINTER. In addition, review the API functions in your code that accept structures as input to ensure that those that contain string or string pointer members are declared as Unicode as appropriate, if the API requires it.

Update your script to call W versions of Win32 APIs if it is currently calling A versions.

Note that now that the InstallScript engine has been updated to support Unicode, you may see changes in the return values for the InstallScript function StrLength. As a result, you may need to make changes to your InstallScript code calls to this function.

StrLength now behaves the same as **StrLengthChars**; both of these functions return the number of characters in a given string variable (that is, the number of code units in the UTF-16-encoded string) up to the first null character. This applies to all InstallScript projects that are created in InstallShield 2016, as well as all InstallScript projects that you have upgraded from earlier versions of InstallShield to InstallShield 2016. Previously, **StrLength** returned the number of characters in a given string variable before a null character in the ANSI code page. The behavior of **StrLengthChars** has not changed.

Differential Release Support in InstallScript Projects

A differential release that is created in an InstallShield 2011 or later InstallScript project can update a product only if its earlier InstallScript installation was also created in InstallShield 2011 or later. If you want to use InstallShield 2011 or later to create an update for a product whose earlier InstallScript installation was created with InstallShield 2010 or earlier, you should create a full release, instead of a differential release.

This requirement is necessary because of the Unicode support that is new in the InstallShield 2011 InstallScript engine. After a differential release is used to update an earlier version of a product on a target system, the earlier version of the installation (along with the earlier engine run time) is used to run any maintenance operations. Because the InstallShield 2010 and earlier versions of the InstallScript engine cannot read the new Unicode storage format, the installation fails.

Uninstalling 64-Bit Registry Entries that Were Installed by the InstallScript Engine

By default, the InstallScript engine now logs the changes that it makes to the 64-bit part of the registry during an InstallScript or InstallScript MSI installation on 64-bit target systems. Furthermore, the 64-bit registry changes that are logged by the InstallScript engine are now uninstalled during uninstallation.

Note that if you created an InstallShield 2010 or earlier installation that wrote 64-bit registry data and you create an upgrade for your product in InstallShield 2016, the existing 64-bit registry entries that were logged by the base installation are not removed when the product is uninstalled from a target system. The only way to work around this limitation is to manually remove the registry data during uninstallation.

Script Editor Changes

Pressing CTRL+SPACEBAR from within the script editor pane in the InstallScript view no longer displays a list of built-in InstallScript functions that are available for auto completion, as it did in InstallShield 2010 and earlier. Now if you want to see a list of built-in InstallScript functions for auto completion, you simply need to start typing the first letter or letters of that function. The pop-up list also contains other InstallScript keywords. To learn more, see Using Auto Completion when Writing Code in the Script Editors.

The context menu that is displayed when you right-click in a script editor pane is now different than it was in InstallShield 2010 and earlier:

- The context menu no longer contains a Show Whitespace command. To show or hide whitespace characters and symbols in the script editors, use the Script Editor Properties dialog box. This dialog box also lets you modify other settings in the script editors, such as font, syntax colors, and line numbering. To access this dialog box, right-click in a script editor and then click Properties.
- The context menu no longer contains a Make Uppercase command or a Make Lowercase command. To make text in the script editor uppercase, select it and then press CTRL+SHIFT+U. To make it lowercase, select it and then press CTRL+U.
- The context menu no longer contains a Find command or a Replace command. To perform a search in a script, press CTRL+F, and then specify the text that you want to find. To search for a string and replace it with something else, press CTRL+H and then specify the appropriate information. As an alternative, you can use the Find and Replace commands on the Edit menu.

For more keyboard shortcuts, see Keyboard Shortcuts for the Script Editors.

The views that contain the revised script editor are the InstallScript view, the SQL Scripts view, and the Custom Actions and Sequences view (when you are viewing a VBScript or JScript file in this view).

InstallScript Debugger Changes

If you copy the InstallScript Debugger (**ISDbg.exe**) from your installation development machine to a debug machine so that you can debug InstallScript code, you must also now copy the file called **SciLexer.dll** to the same folder on the debug machine. You can find **SciLexer.dll** on your installation development machine in the same folder as the **ISDbg.exe** file (*`InstallShield Program Files Folder`***System**).

For more information, see Debugging an Installation on Any Computer.

Changes that Affect New Projects but Not Upgraded Projects

This section describes changes to InstallShield that may affect new projects but not projects that are upgraded from earlier versions. Note that you may need to make manual changes to upgraded projects.

DPI Changes and Their Effect on InstallScript Dialogs

If you have an InstallShield 2010 or earlier InstallScript or InstallScript MSI project that contains one or more edited InstallScript dialogs and you want to upgrade the project to InstallShield 2016 or later, ensure that your machine uses the same DPI value that was selected when the InstallScript dialogs were edited. Otherwise, the dialogs may be sized incorrectly at run time.

Note that once you have upgraded the project, you can change the DPI value on your machine at any time as needed; if you do, the dialogs are sized correctly at run time. Also note that if you upgrade an InstallShield 2016 or later project that contains one or more edited InstallScript dialogs to a future version of InstallShield, you do not need to use the same DPI value during the upgrade.

Changing Design-Time and Build-Time Locations of Existing InstallShield Prerequisites in Existing Projects

InstallShield now lets you specify the folders where InstallShield should search for InstallShield prerequisite files (.prq files), their associated data files, and their dependencies. Previously, InstallShield searched for .prq files in the following location only: *InstallShield Program Files Folder***SetupPrerequisites**.

If you move any InstallShield prerequisites from the *InstallShield Program Files Folder***SetupPrerequisites** folder to a new custom location that you have defined on the Prerequisites tab of the Options dialog box (or any of the other places where search paths can be defined now), you may need to perform the following steps in InstallShield 2010 or earlier projects when you upgrade them to InstallShield 2016:

1. In the Redistributables view or the Prerequisites view, clear the check box for each InstallShield prerequisite that is included in your project but is located in a custom location. Also clear the check box for each InstallShield prerequisite whose data files or dependencies were moved from the default location to a custom location.
2. Click the new Refresh button.
3. Select the check box for each InstallShield prerequisite that you removed from your project in step 1.

InstallShield removes the path of the prerequisite from the **ISSetupPrerequisites** table of your project. The full path was stored in this table in InstallShield 2010 and earlier projects. Note that if you only clear a prerequisite's check box and then reselect it without clicking the Refresh button, InstallShield continues to use the full path, rather than just the file name, in the **ISSetupPrerequisites** table.

If you upgrade an InstallShield 2010 or earlier project to InstallShield 2016, change the location of an InstallShield prerequisite, and then add that prerequisite to your project, you do not need to perform the refresh procedure. Also, if you create a new project in InstallShield 2016, you do not need to perform the refresh procedure. In both cases, InstallShield does not include the path in the **ISSetupPrerequisites** table of your project, which enables you to use the custom search path, instead of the default path.

Preventing an InstallScript MSI Installation from Overwriting a Future Major Version of the Same Product

The Upgrades view in new InstallScript MSI projects now contains a major upgrade item called ISPreventDowngrade. This item prevents end users from being able to install the current version of your product over a future major version of the same product. If you upgrade an InstallScript MSI project from InstallShield 2010 or earlier to InstallShield 2016, the ISPreventDowngrade item is not added automatically. You can manually add an ISPreventDowngrade item if appropriate. To learn how, see the Preventing the Current Installation from Overwriting a Future Major Version of the Same Product.

Upgrading Projects from InstallShield 2009 or Earlier

The following information describes possible upgrade issues that may occur when you upgrade projects that were created with InstallShield 2009 and earlier to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from InstallShield 2009 or earlier to InstallShield 2016.

General Information about Upgrading Projects that Were Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open a project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .768 before converting it. Delete the .768 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2016: InstallShield 2009 and earlier, InstallShield 12 and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2016.

Installing More than One Edition of InstallShield

Only one edition of InstallShield 2016—Premier, Professional, or Express—can be installed on a system at a time. Previously, it was possible to install the Express edition on the same system that had the Premier or Professional edition of the same InstallShield version.

Change to the List of Supported Operating Systems for Running InstallShield

The minimum operating system requirement for systems that run InstallShield (the authoring environment) is now Windows XP or Windows Server 2003. Previously, the minimum operating system requirement was Windows 2000 SP3.

Changes that Affect All Projects (New and Upgraded Projects)

This section describes changes that affect both new projects and projects that are upgraded from earlier versions of InstallShield.

Setup.exe No Longer Runs on Windows 9x, Windows NT 4, or Windows Me Systems

Setup.exe installations that are created in InstallShield can no longer be run on Windows 9x, Windows NT 4, or Windows Me. If an end user tries to launch **Setup.exe** on a Windows 9x or Windows Me system, Windows displays a message box with the following error: "The *FullSetup.exePathAndFileName* file expects a newer version of Windows. Upgrade your Windows version." On Windows NT 4 systems, Windows displays a message box with the following error: "*FullSetup.exePathAndFileName* is not a valid Windows NT application."

InstallShield no longer lists these legacy operating systems in any of the areas where target operating systems can be selected. For example, the Installation Requirements tab of the Project Assistant in Basic MSI and InstallScript MSI projects no longer lists these operating systems. In InstallScript projects, the Platforms tab on the Project Settings dialog box no longer lists these operating systems.

If you upgrade an InstallScript project that was created in InstallShield 2009 or earlier to InstallShield 2016, and if the operating system settings in the earlier project contained references to only these legacy operating systems, InstallShield replaces the legacy operating system options with the option for targeting all supported platforms.

Windows Installer 1.x Redistributables Are No Longer Available

InstallShield no longer includes Windows Installer 1.x redistributables, since they target only legacy versions of Windows that are no longer supported. Previously, it was possible to add Windows Installer 1.x redistributables to a project through the Releases view, or through the Release Wizard.

Redistributable for VBScript Runtime Files Is No Longer Available

InstallShield no longer includes the InstallShield object for VBScript Runtime Files. This redistributable targets only legacy versions of Windows that are no longer supported.

InstallScript Dialog Source Code Changes

The InstallScript code for built-in InstallScript dialogs has been moved from individual InstallScript script files (.rul) to a single **ISRTScriptDialogs.rul** file. In addition, the event class drop-down list in the InstallScript view has a new Dialog Source option. If you select this option, the event handler drop-down list shows all of the built-in InstallScript dialogs. You can select any dialog in this list to customize its code.

InstallShield supports backwards compatibility: If you imported dialog source code into an InstallShield 2009 or earlier project and then you upgrade the project to InstallShield 2016, you can still use that dialog code. However, if you want to use the dialog sources that are now available when you select the dialogs in the event handler drop-down list, you must first make some changes to your upgraded project. Otherwise, you may encounter compile errors. In order to use the dialog code: On the Build menu, click Settings. On the Compile/Link tab, in the Preprocessor Defines box, enter the following:

_ISSCRIPT_NEW_STYLE_DLG_DEFS

After this definition has been added, if any dialogs had previously been imported into the project in earlier versions of InstallShield, the code in these imported .rul files may cause compile errors. These errors would need to be resolved.

If _ISSCRIPT_NEW_STYLE_DLG_DEFS is not defined, a warning message is displayed when the Dialog Source option is selected in the event class drop-down list in the InstallScript view.

The _ISSCRIPT_NEW_STYLE_DLG_DEFS definition is automatically added to all new projects that are created in InstallShield 2016.

This functionality applies to the following project types: InstallScript, InstallScript MSI, and InstallScript Object.

InstallScript Header File Changes

The InstallScript header files (.h) have been reorganized. As a result, some of the .h files are obsolete. If an InstallScript file (.rul) in your project references one or more of these obsolete .h files, a compile warning is displayed whenever you compile your script or build a release. To resolve the warnings, remove any #include statements that reference the obsolete .h files, and rebuild the release. Also, ensure that **ifx.h** is referenced in an #include statement in your **Setup.rul** file or in other script files that are referenced by **Setup.rul**.

InstallScript Changes for Windows API Prototypes

The service-related Windows API prototypes (such as CreateServiceA, StartServiceA, and ControlService) are now prototyped in **ISRTWindows.h** for all InstallScript, InstallScript MSI, and InstallScript Object projects. They are also now prototyped in **ISRTWindows.h** for all Basic MSI and Merge Module projects that include InstallScript custom actions.

If you want to use your own definitions for these Windows APIs instead of the ones that are now prototyped for InstallScript, add **ISINCLUDE_NO_SERVICEAPI** to the list of preprocessor definitions: On the Build menu, click Settings. On the Compile/Link tab, in the Preprocessor Defines box, enter **ISINCLUDE_NO_SERVICEAPI**. Otherwise, you may encounter compile errors.

Note that the constant ISINCLUDE_NO_WINAPI_H now suppresses only Windows API prototypes, not Windows constants or Windows structure definitions.

Patch Creation (Basic MSI, InstallScript MSI)

InstallShield now uses the Windows Installer 4.5 patching technology to create patches. This change is reported for informational purposes.

New Custom Actions and Database Tables for IIS Support in Basic MSI and InstallScript MSI Projects

If you add IIS support through the Internet Information Services view in a Basic MSI or InstallScript MSI project in InstallShield, InstallShield automatically adds several DLL custom actions to your project to support the IIS functionality. In InstallShield 2016, these custom actions have been enhanced to enable you to add applications to Web sites. In addition, these actions have been renamed to reflect standard naming conventions:

- **ISIISCosting**—This custom action replaces the caExtractIISSuppFiles action.
- **ISIISRollback**—This custom action replaces the caRlbackVRoots action.
- **ISIISUninstall**—This custom action replaces the caRemoveVRoots action.
- **ISIISInstall**—This custom action replaces the caCreateVRoots action.
- **ISIISCleanup**—This custom action replaces the calISCleanup action.

The entry points of each of the DLL custom actions have been renamed to match the corresponding custom action names.

Note that IIS functionality requires administrative privileges. Therefore, each of these DLL custom actions checks the Privileged property value. The value must be 1; if it is not, an error is displayed at run time. In InstallShield 2009 and earlier, each of the IIS custom actions had a Privileged = 1 condition. In InstallShield 2016, this condition is no longer set when you add new IIS Web sites or other IIS data to your project, since the custom actions check the Privileged property value.

If you upgrade a Basic MSI or InstallScript MSI project that contains IIS support from InstallShield 2009 or earlier to InstallShield 2016, InstallShield automatically updates and renames the custom actions accordingly. In addition, if the condition for an old custom action was not modified from the default condition, InstallShield also removes the Privileged = 1 condition. If a condition was modified, InstallShield leaves the existing condition as is. You can manually modify any of the conditions if appropriate.

In InstallShield 2016, all of the IIS data is stored in the **ISIISItem** and **ISIISProperty** tables. In InstallShield 2009 and earlier, the IIS data was stored in the following tables: **ISIISAppPool**, **ISIISCommon**, **ISIISMetaData**, **ISIISWebServiceExtension**, **ISVRoot**, **ISVRootAppMaps**, and **ISWebSite**. If you upgrade a Basic MSI or InstallScript MSI project that contains IIS support from InstallShield 2009 or earlier to InstallShield 2016, InstallShield automatically moves the IIS data to the new tables; InstallShield also deletes the old tables from the project.

For more information, see:

- Managing Internet Information Services
- InstallShield Custom Action Reference

Changes for the Redistributables View

The Redistributables view has a new toolbar and group box area that provide robust search and organizational functionality. Use the new Show Details button in this view to show or hide the details pane for the selected redistributable in this view. The details pane provides information such as which files a redistributable installs. The Show Details button replaces the Show Details and Hide Details links that were previously available in the upper-right corner of this view.

The new group box area is below the new toolbar in the Redistributables view. You can drag and drop column headings onto this group box area to organize the list of redistributables in a hierarchical format. If you want InstallShield to separate all of the redistributables in the view into two groups—one whose check box is selected and one whose check box is cleared—drag the check box column to the group box area. This enables you to easily identify all of the redistributables that are included in your project. The result is similar to the behavior that previously occurred if you right-clicked any redistributable and then clicked Show Only Selected Items. Note that the Show Only Selected Items command is no longer available in the Redistributables view.

For more information, see Working with the Group Box Area in Various Views.

Limiting the UI Level of a Chained .msi Package to that of the Main .msi Package

The UI level for a chained .msi package is now limited to be no higher than that of the parent package's current UI level. For example, in the following scenario, the chained .msi package is launched silently: you add a chained .msi package to your project in the Releases view and select Full UI (/qf) for its UI level setting, but the main installation is launched silently (/qn). Previously, the chained package showed exactly the UI level that was authored in the Releases view of the main installation; to restore this behavior, set the property **ISChainExceedUILevel** equal to the value 1.

Encoding and Related Differences for XML File Changes

If you use the XML File Changes view to configure changes for a file that is already present on the target machine, or that is being installed as part of your installation, the installation now uses the encoding that is specified in that XML file, rather than the encoding that is specified in the XML File Changes view. This applies to new projects that are created in InstallShield 2016, as well as projects that are upgraded from InstallShield 2009 or earlier.

In addition, if the "Always create this element if it does not already exist" check box is cleared for an element that is not present in the target file, its child elements are no longer created. Thus, for an XML file such as **//A/B/C**, C is not created on the target system if B is neither present nor set to be created.

Changes to the Major and Minor Version Registry Entries for the Uninstall Key of InstallScript Installations

InstallScript installations now always create VersionMajor and VersionMinor registry values in the Uninstall key; the names of these values now match the entries that are created during Basic MSI and InstallScript MSI installations. This applies to new installations that are created in InstallShield 2016, as well as installations that are upgraded from InstallShield 2009 or earlier. Previously, in InstallShield 2009 and earlier, the names of the values that InstallScript installations created were MajorVersion and MinorVersion; these are no longer created.

In order to use the new registry values, the values of the following InstallScript constants have been changed:

- REGDB_VALUENAME_UNINSTALL_MAJORVERSION is now VersionMajor instead of MajorVersion.
- REGDB_VALUENAME_UNINSTALL_MINORVERSION is now VersionMinor instead of MinorVersion.

When the **MaintenanceStart** function is called, it creates the updated value names in the registry. By default, it also deletes the old value names if they exist. If you do not want the old value names to be deleted from target systems, you can use the new REGDB_OPTIONS option called REGDB_OPTION_NO_DELETE_OLD_MAJMIN_VERSION. If you want to continue using only the old value names, you must delete the new versions after **MaintenanceStart** returns.

If REGDB_UNINSTALL_MAJOR_VERSION or REGDB_UNINSTALL_MINOR_VERSION is used with the **RegDBGetItem** function, **RegDBGetItem** first checks for the new value; if the new value is found, the function returns the value data from the new value. If the new value is not found, the function automatically checks for the old value; if the old value is found, the function returns the value data from the old value.

To provide backwards compatibility, the following new constants are available:

- REGDB_UNINSTALL_MAJOR_VERSION_OLD
- REGDB_UNINSTALL_MINOR_VERSION_OLD

You can specify these constants with the **RegDBGetItem**, **RegDBSetItem**, and **RegDBDeleteItem** functions to get, set, and delete the old values.

The following new string constants are also available:

- REGDB_VALUENAME_UNINSTALL_MAJORVERSION_OLD is defined as MajorVersion.
- REGDB_VALUENAME_UNINSTALL_MINORVERSION_OLD is defined as MinorVersion.

For more information, see the following:

- REGDB_VALUENAME_UNINSTALL_MAJORVERSION
- REGDB_VALUENAME_UNINSTALL_MAJORVERSION_OLD
- REGDB_VALUENAME_UNINSTALL_MINORVERSION
- REGDB_VALUENAME_UNINSTALL_MINORVERSION_OLD
- REGDB_UNINSTALL_MAJOR_VERSION_OLD
- REGDB_UNINSTALL_MINOR_VERSION_OLD
- REGDB_OPTIONS
- RegDBSetItem
- RegDBGetItem
- RegDBDeleteItem

Removal of the SdShowMsg Dialog from the List of Editable Dialogs

The Dialog Editor does not currently support dialogs, such as **SdShowMsg**, that do not have a title bar; if you try to customize the **SdShowMsg** dialog, it may get corrupted. Therefore, this dialog is no longer displayed in the Dialogs view as one of the dialogs that you can edit. To customize this dialog, you should use the **SdShowMsg** call, not the Dialog Editor.

Automation Interface Changes

The value of the eosAll constant for the OSFilter, which is a member of the ISWiComponent and ISWiRelease objects in the automation interface, has been changed. The new value is 64028880; previously, it was 5308624. If you are using the value of this constant to configure the list of operating systems for a component or a release through the automation interface, you must update your script to use the new value.

For more information, see the following:

- ISWiComponent Object
- ISWiRelease Object

Changes for the Locations of InstallScript Run-Time Script, Library, and Header Files

The InstallScript run-time library files that are installed with InstallShield have been consolidated into a central location, instead of several separate subdirectories. The script, library, and header files are now installed in Src, Lib, and Include folders in the following directory:

*InstallShield Program Files Folder***Script\Isrt**

The following folders are no longer installed, since the files within them are consolidated in the above location:

*InstallShield Program Files Folder***Script\IISRuntime**

*InstallShield Program Files Folder***Script\SQLRuntime**

*InstallShield Program Files Folder***Script\XMLRuntime**

Note that because of a name conflict, the **Assert.h** file in the following location is being renamed as **ISAssert.h**:

*InstallShield Program Files Folder***Script\Isrt\Include**

If you create a new project in InstallShield 2016, it uses the new locations of the files. If you upgrade a project from InstallShield 2009 or earlier to InstallShield 2016, InstallShield updates the projects to use the new locations.

Changes in the Way that Linked Libraries and Their Locations Are Specified

The Compile/Link tab on the Settings dialog box has a new Additional Library Paths box that lets you specify the locations where the InstallScript compiler should search for InstallScript libraries (.obl files) that are not one of the standard InstallShield locations. For InstallScript and InstallScript Object projects, the standard locations are:

- **<ISProductFolder>\Script\Ifx\Lib**
- **<ISProductFolder>\Script\Isrt\Lib**

For Basic MSI and InstallScript MSI projects, the standard locations are:

- **<ISProductFolder>\Script\Iswi\Lib**
- **<ISProductFolder>\Script\Isrt\Lib**

If you create a new project in InstallShield 2016, InstallShield automatically lists the standard InstallShield script libraries such as **ISRT.obl** in the Libraries (.obl) box on the Compile/Link tab. However, InstallShield no longer includes the full path in that box. If you want to add your own custom libraries, you can specify the library file name in the Libraries (.obl) box, and the path in the Additional Library Paths box. You do not need to specify the full path and file name in the Libraries (.obl) box.

If you upgrade a project that was created in InstallShield 2009 or earlier to InstallShield 2016, InstallShield automatically removes the path of the standard script libraries that are listed in the Libraries (.obl) box. For more information, see Compile/Link Tab.

Removal of the InstallScript Structure Definition for ISOSVERSIONINFO

The definition of the ISOSVERSIONINFO structure, as well as the corresponding unused global instances of this structure, has been removed. The equivalent OSVERSIONINFO structure is still available. This definition removal does not cause any functionality changes; however, if you attempt to use the ISOSVERSIONINFO structure definition or global structure instances, a compiler error results.

To avoid a compiler error, do either of the following:

- Update the script to use the equivalent OSVERSIONINFO structure, and declare a local instance of this structure if needed. Update the script to use the appropriate structure member names. (Note that the OSVERSIONINFO member names are different than the ISOSVERSIONINFO member names.) Following is the definition of OSVERSIONINFO:

```
typedef OSVERSIONINFO
    begin
        NUMBER nOSVersionInfoSize;
        NUMBER nMajorVersion;
        NUMBER nMinorVersion;
        NUMBER nBuildNumber;
        NUMBER nPlatformId;
        STRING szCSDVersion[128];
    end;
```

- Declare the structure and structure instances locally as follows:

```
// Data structure that contains operating system version information.
// Used by ISCompareServicePack.
typedef ISOSVERSIONINFO // define a structure
begin
    LONG ISIOSVersionInfoSize;    // Size in bytes of this data structure
    LONG ISIMajorVersion;         // Major version number of the OS.
    LONG ISIMinorVersion;         // Minor version number of the OS.
    LONG ISIBuildNumber;          // Build number of the OS.
    LONG ISIPlatformId;           // Operating system platform.
    STRING szISCSDVersion [128];  // Additional information about OS.
end;

// Variable for the operating system version information data structure.
// Used by ISCompareServicePack.
ISOSVERSIONINFO ISVersion;

// Pointer that points to the OS version information variable.
// Used by ISCompareServicePack.
ISOSVERSIONINFO POINTER pISVersion;
```

Obsolete Keys Are No Longer Written to Setup.ini for InstallScript Projects

The following keys are obsolete and are no longer written to the **Setup.ini** file for InstallScript projects: Resource, EngineVersion, and EngineBinding.

ISCab.exe Is No Longer Available

ISCab.exe is no longer supported. Therefore, it is no longer included with InstallShield.

Changes that Affect New Projects but Not Upgraded Projects

This section describes changes to InstallShield that may affect new projects but not projects that are upgraded from earlier versions. Note that you may need to make manual changes to upgraded projects.

Changes to Support for Securing Permissions for Files, Folders, and Registry Keys

The General Information view has a new Locked-Down Permissions setting that lets you specify whether you want to use the new custom InstallShield handling or the traditional Windows Installer handling for all new permissions that you set for files, folders, and registry keys in your project. The new custom InstallShield handling option offers several advantages over the traditional Windows Installer handling option.

In all new projects, the default value for this setting is the custom InstallShield handling option. If you upgrade a project from InstallShield 2009 or earlier to InstallShield 2016, the traditional Windows Installer handling option is the default value of this setting.

This new setting is available in the following project types: Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform.

For more information, see the following:

- Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment
- Selecting the Locked-Down Permissions Type for a Project

Changes to the ReadyToInstall Dialog for Beta Windows Installer 5 Support of Per-User Installations

The General Information view has a new Show Per-User Option setting. This setting lets you specify whether you want the ReadyToInstall dialog—in certain scenarios—to include buttons that let end users indicate how they want to install the product: for the current user or for all users. The per-user button sets the new Windows Installer property **MSIINSTALLPERUSER** equal to 1 to indicate that the package should be installed for the current user. The **MSIINSTALLPERUSER** property is available with the beta of Windows Installer 5.

If you create a new Basic MSI project in InstallShield 2016, the ReadyToInstall dialog includes support for the per-user and per-machine buttons; these buttons are displayed or hidden at run time if appropriate. If you upgrade a Basic MSI project from InstallShield 2009 or earlier to InstallShield 2016, the ReadyToInstall dialog does not have this support automatically. You can manually add these buttons and their associated conditions to the ReadyToInstall dialog if appropriate; use the ReadyToInstall dialog in a new InstallShield 2016 project as a guideline.

Public Directory Properties for Feature Destinations Are Added to SecureCustomProperties

When you specify a location for the Destination setting of a feature and the location includes a public directory property, InstallShield now adds that property to the **SecureCustomProperties** property to allow end users to change the destination after the product has been advertised. This occurs in new projects that are created in InstallShield 2016. The change is also made for all feature destinations if you upgrade a project from InstallShield 2009 or earlier to InstallShield 2016.

This change applies to the following project types: Basic MSI and InstallScript MSI.

Changes to the Conditions for the InstallWelcome Dialog and the ResolveSource Action

The condition on the InstallWelcome dialog and the ResolveSource action has been changed to **Not Installed** for all new Basic MSI projects that are created in InstallShield 2016. The conditions were changed so that the InstallWelcome dialog and the ResolveSource action can be used for a first-time installation with a patch. If you upgrade a Basic MSI project from InstallShield 2009 or earlier to InstallShield 2016, the conditions are not changed automatically. If you want the dialog and action to be used for a first-time installation with a patch, you can change the conditions in your upgraded project to **Not Installed**.

Improvements to the .rtf File Size Limit for the SdLicenseRtf and SdLicense2Rtf Functions

The file size limit for the .rtf files that are used with the InstallScript dialog functions **SdLicenseRtf** and **SdLicense2Rtf** is now 16 MB instead of 64 KB. Previously, if the file size was more than 64 KB, part of the EULA text was missing from the license dialog at run time.

Note that if you had overridden the **SdLicenseRtf** or **SdLicense2Rtf** functions in your script in InstallShield 2009 or earlier and then upgraded that project to InstallShield 2016, you would need to manually change the size limit by updating the **SendMessage** call with the EM_EXLIMITTEXT message in DLG_INIT. The iParam parameter (the fourth parameter) of the **SendMessage** call needs to be changed. The **SendMessage** call should be changed to this:

```
SendMessage( hEdit, EM_EXLIMITTEXT, 0, 0xffffff );
```

Previously, the code contained this:

```
SendMessage( hEdit, EM_EXLIMITTEXT, 0, 0 );
```

Removal of the OnResolveSource Event Handler from InstallScript MSI Installations

The InstallScript event handler OnResolveSource has been removed from InstallScript MSI projects. The Windows Installer now handles all source resolution. If you added the OnResolveSource event to an InstallScript MSI project in InstallShield 2009 or earlier and then you upgrade that project to InstallShield 2016, that event will no longer be called.

Changes to the Way that a Log File Is Displayed from the SetupCompleteSuccess Dialog in Basic MSI Installations

The ShowMsiLog custom action now launches **Notepad.exe** from the SystemFolder directory, instead of from the WindowsFolder directory. Thus, if your installation is run on Windows Vista or later and the end user indicates on the SetupCompleteSuccess dialog that they want to view the log file, the installation launches **Notepad.exe** from the SystemFolder directory. This change was made because on Windows Server 2008 Standard Edition, **Notepad.exe** is available in the System32 directory, but not the Windows directory.

Note that behavior is available by default in all new Basic MSI projects that are created in InstallShield. If you upgrade an InstallShield 2009 or earlier Basic MSI project to InstallShield 2016, InstallShield does not automatically change the behavior. You can manually change the behavior if necessary: In the Custom Actions and Sequences view, click the ShowMsiLog action. (If this action is not displayed, right-click the Custom Actions node and then click Show All Custom Actions.) Set the Filename & Commandline setting as follows:

[SystemFolder]notepad.exe "[MsiLogFileLocation]"

Thus, the value should contain [SystemFolder] instead of [WindowsFolder].

Changes for ALLUSERS Property in InstallScript MSI Installations

Beginning with InstallShield 2010, the **ALLUSERS** property is set to 1 by default in all new InstallScript MSI projects. This is the recommended implementation, since most installations must be run in a per-machine context with administrative privileges. This value is also recommended to help avoid **ALLUSERS**-related issues when an InstallScript MSI installation is run silently.

If you upgrade a project that was created with InstallShield 2009 or earlier to InstallShield 2016, InstallShield does not automatically change the value of the **ALLUSERS** property or add this property if it was not defined in the earlier project.

InstallScript Installations No Longer Include _Setup.dll

InstallScript installations no longer include **_Setup.dll**. Some earlier versions (DevStudio 9 and InstallShield X) did not log **_Setup.dll** for uninstallation. As a result, this file was left behind in the Disk1 folder location (DISK1TARGET) after uninstallation. If an update was created with a later InstallShield version (InstallShield 10.5 through InstallShield 2009) and the update was for an original installation that was created with DevStudio 9 or InstallShield X, **_Setup.dll** was deleted during uninstallation because **_Setup.dll** was logged by the update. Since **_Setup.dll** is not included in InstallScript installations that are created with InstallShield 2016, the **_Setup.dll** file may now be left behind. Therefore, if you are updating from an installation that was created with DevStudio 9 or InstallShield X, you may need to delete the **_Setup.dll** file manually (DISK1TARGET ^ "_Setup.dll") during uninstallation to ensure that the uninstallation is complete.

Saving a Project as an Earlier Version

InstallShield no longer has support for downgrading a project. That is, you cannot save an InstallShield 2016 project as an InstallShield 2009 or earlier project.

Trialware Support

The only edition of InstallShield that includes the Trialware view is the Premier edition. This edition lets you create the Try and Die type of trialware. InstallShield no longer includes support for creating the Try and Buy/Product Activation type of trialware.

Web Projects

The Web project type is no longer listed as one of the types of new projects that you can create in InstallShield. To use the same functionality that was available with a Web project in InstallShield 2009 and earlier, create a Basic MSI project, and then add a Web site in the Internet Information Services view.

The only difference between a Web project and a Basic MSI project was that a new Web project automatically contained a predefined folder for the **IISROOTFOLDER** directory in the Files and Folders view. All of the files that you add to the **IISROOTFOLDER** directory are installed to the Web server's root directory on the target system. InstallShield adds predefined folder for the IISROOTFOLDER directory to a Basic MSI project when you add a Web site to the project. Thus, a Basic MSI project that contains at least one Web site configured in the Internet Information Services view is equivalent to a Web project that was created in InstallShield 2009 or earlier.

Compact Projects

InstallShield no longer has support for Compact projects.

Visual Studio Integration

Microsoft Visual Studio can be integrated with only one version of InstallShield Premier Edition or InstallShield Professional Edition at a time. The last version of InstallShield that is installed or repaired on a system is the one that is used for Visual Studio integration.

Upgrading Projects from InstallShield 2008 or Earlier

The following information describes possible upgrade issues that may occur when you upgrade projects that were created with InstallShield 2008 and earlier to InstallShield 2016. It also alerts you to possible changes in behavior that you may notice between new InstallShield 2016 projects and projects that are upgraded from InstallShield 2008 or earlier to InstallShield 2016.

General Information about Upgrading Projects that Were Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open a project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .766 before converting it. Delete the .766 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2016: InstallShield 2008, InstallShield 12 and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2016.

Changes that Affect All Projects (New and Upgraded Projects)

This section describes changes that affect both new projects and projects that are upgraded from earlier versions of InstallShield.

New Default Setup Launcher Value for New Releases: Windows Installer Is Not Included

When you create a new release in a Basic MSI or InstallScript MSI project, the redistributable for the Windows Installer engine is no longer included by default:

- If you use the Release Wizard to create a new release, the Setup Launcher panel is where you specify whether to include the Windows Installer. The default value for the **Support these operating systems** setting is **Do not install Windows Installer**. Previously, the default value was **Both Windows 9X and NT**.
- If you create a new release by right-clicking the Releases explorer in the Releases view, the default value for the Setup Launcher setting on the Setup.exe tab is now **Yes (no MSI engine included)**. Previously, the default value for this setting was **Yes (include Windows NT & Windows 9x MSI engine)**.

This change applies to all new releases that are created in new InstallShield 2016 projects.

If you upgrade a project from InstallShield 2008 or earlier to InstallShield 2016, this change applies to all new releases; InstallShield does not automatically change the value for releases that were originally created in the earlier InstallShield version.

Upgrade and Patch Validation

When you build QuickPatch projects, InstallShield now runs patch and upgrade validation. Therefore, when you build a QuickPatch package in InstallShield 2016, you may see validation errors and warnings. If you built a QuickPatch project in InstallShield 2008 or earlier, InstallShield did not run the upgrade and patch validation; thus, patch or upgrade validation errors or warnings were never displayed at build time.

In addition, the Val0015 warning now checks the **ISSelfReg** table in QuickPatch projects and in patches that are created through the Patch Design view in Basic MSI and InstallScript MSI projects. If a QuickPatch or patch project adds a row to the **ISSelfReg** table, Val0015 warns you that the patch will be uninstallable. Previously, Val0015 did not check for entries in this table.

File Compression for Files that Are Streamed into Setup.exe and ISSetup.dll at Build Time

If you build a release that uses a **Setup.exe** setup launcher or a **ISSetup.dll** file, InstallShield now compresses files that it streams into the **Setup.exe** file or the **ISSetup.dll** file at build time. The default compression level that InstallShield uses offers a balance between file size and time that is required to extract the compressed files at run time. This applies to new Basic MSI and InstallScript MSI projects as well as existing Basic MSI and InstallScript MSI projects that are upgraded from InstallShield 2008 or earlier to InstallShield 2016.

If you want to change the compression level or you do not want to use any compression, you can override the default level through a machine-wide setting. For more information, see Configuring the Compression Level for Files that Are Streamed into Setup.exe and ISSetup.dll.

Previously, InstallShield did not include any support for compressing files that were streamed into the **Setup.exe** file or the **ISSetup.dll** file at build time. Thus, if you compare a release that was built in InstallShield 2008 or earlier with the same release that is built with the default compression level in InstallShield 2016, you may notice that the file size of **Setup.exe** or **ISSetup.dll** is slightly different. In addition, the time that is required to extract files may be slightly different.

Multi-Part .cab Files

InstallShield now has a default limit of 600 MB for each .cab file that it creates at build time for a network image release in which the compression type is the standard compression type (not custom compression) and all of the files are embedded in a single-file .msi package or a **Setup.exe** setup launcher. When InstallShield is creating the .cab files for this type of release and it reaches this limit, it splits the data into two or more .cab files, creating multi-part .cab files. This applies to new Basic MSI and InstallScript MSI projects as well as existing Basic MSI and InstallScript MSI projects that are upgraded from InstallShield 2008 or earlier to InstallShield 2016.

You can modify the .cab size limit if necessary. In addition, if you do not want InstallShield to create multi-part .cab files, you can configure it to create single .cab files. For more information, see Configuring the Maximum Size for .cab Files.

Previously, InstallShield did not create multi-part .cab files, and there was no built-in limit for the .cab file size.

Automation Interface and Standalone Build

If you have existing automation scripts that work with the InstallShield Automation Interface, you no longer need to change the library name from IswiAuto*N* to SAAuto*N* (where *N* indicates the version number) throughout the scripts in order to use them with the Standalone Automation Interface.

The Standalone Build that is available with InstallShield Premier Edition now uses the same directory structure that InstallShield uses.

The **ISCmdBld.exe** file that is used for command-line builds with InstallShield is now installed with the Standalone Build. Previously, the Standalone Build used **IsSaBld.exe**, a different file, for command-line builds.

FlexNet Connect Support in InstallScript Projects

InstallScript projects no longer include an Update Notifications view, which is used to add FlexNet Connect support to an installation. Note that this view is still available in Basic MSI and InstallScript MSI projects.

Note the following details about FlexNet Connect support in InstallScript projects:

- InstallShield does not add **ISUS.obl** to the list of libraries (or enable any other FlexNet Connect functionality) when ISEnableUpdateService is changed to 1 in the **InstallShield** table. The **ISUS.obl** library is no longer supported.
- If you upgrade an InstallScript project from InstallShield 2008 or earlier to InstallShield 2016, InstallShield automatically disables FlexNet Connect support and removes the **ISUS.obl** from the list of linked script libraries.
- The InstallScript constants and functions for FlexNet Connect still compile. However, all of the functions—except for **UpdateServiceGetAgentTarget**—now return ISERR_NOT_IMPLEMENTED. **UpdateServiceGetAgentTarget** returns null ("").
- All calls to the InstallScript functions for FlexNet Connect were removed from the default InstallScript code. In the OnUpdateUIBefore event handler, the code for **UpdateServiceOnEnabledStateChange** was removed. In the OnMoveData event handler, the code blocks for **UpdateServiceRegisterProduct** and **UpdateServiceCreateShortcut** were removed.

  Note that if you upgrade an InstallScript project from InstallShield 2008 or earlier to InstallShield 2016 and you had overridden these events in the earlier version of your project, the aforementioned functions are called. This should not cause any problems. The functions now return ISERR_NOT_IMPLEMENTED.

  Also, the OnFirstUIAfter event handler was updated to not include the option of calling **SdFinishUpdate**, which was disabled previously by default.
- All FlexNet Connect–related run-after-reboot functionality was removed. Therefore, the product is not registered with FlexNet Connect after a reboot.
- When you build an InstallScript project, InstallShield no longer adds any FlexNet Connect–related files to the media.
- ISUS.obl will no longer be supported and thus will not be provided with the product.

FlexNet Connect Support in InstallScript MSI Projects

The following FlexNet Connect–related functions have been deprecated for InstallScript MSI projects:

- **GetUpdateStatus**—This function always returns FALSE.
- **SetUpdateStatus**—This function returns ISERR_NOT_IMPL.
- **GetUpdateStatusReboot**—This function always return FALSE.
- **SetUpdateStatusReboot**—This function returns ISERR_NOT_IMPL.

All calls to the **SetUpdateStatus** and **SetUpdateStatusReboot** functions were removed from the default InstallScript code.

Multilanguage Support in Basic MSI and InstallScript MSI Projects

If an installation includes multilanguage support and end users launch Setup.exe again (after the product is already installed) to run the installation in maintenance mode, the language selection dialog is no longer displayed. The maintenance dialogs are displayed in the same language that was used to originally install the product.

Similarly, if a multilanguage minor upgrade or small update is run to update an earlier version of an already installed product, the language selection dialog is not displayed. The upgrade dialogs are displayed in the same language that was used to originally install the product.

This behavior helps ensure that the same language that was used to run the original installation and install its features is also used to repair the original installation, as well as to add other features that were not originally installed but are installed during maintenance mode or upgrades.

InstallScript Multi-Instance Support

For an InstallScript multilanguage, multi-instance installation that displays the language selection dialog, the instance selection dialog is now displayed before the language selection dialog. The dialog order change enables end users to select a different language for a new instance of the product. If end users specify in the instance selection dialog that they want to maintain an existing instance, the language selection dialog is not displayed. Note that when the instance selection dialog is displayed, it is displayed in the same language that would be used to display the language selection dialog.

If multiple instances of a non-multi-instance installation are installed through the **Setup.exe** `/ig` command-line parameter, the language selection dialog is displayed only during a first-time installation; it is not displayed during maintenance mode. This helps ensure that the same language that was used to run the original installation and install its features is also used to repair the original installation, as well as to add other features that were not originally installed but are installed during maintenance mode.

During an InstallScript full-release installation that does not display the instance selection dialog—such as an installation that is run in silent mode or with the **Setup.exe** `-hide_usd` command-line parameter—the installation automatically installs a new instance. This behavior occurs for all new InstallShield 2016 projects, as well as projects that are upgraded from earlier versions to InstallShield 2016. In InstallShield 2008, `/hide_usd` resulted in the first installed instance being maintained, and silent mode resulted in a new instance being installed. In InstallShield 12 and earlier, the first installed instance would be maintained in both the `/hide_usd` and silent mode scenarios.

For a differential media, the first instance read is updated.

During a non-multi-instance installation on a system where two or more instances are installed (using the **Setup.exe** `/ig` command-line parameter), the instance selection dialog is displayed. The instance selection dialog does not allow a new instance to be installed during a non-multi-instance installation because a non-multi-instance installation does not generate a new random GUID for the new instance. The instance selection dialog includes the "default" instance (the instance whose GUID matches the product code) if it installed and it is one of the instances that can be maintained. This behavior applies to all new InstallShield 2016 projects, as well as projects that are upgraded from earlier versions of InstallShield.

If a single instance is installed, the installation automatically attempts to maintain the installed instance; it does not show the instance selection dialog. Note that is true even if the installed instance is not the default instance. As a result of this behavior, if the default instance is not installed but a non-default instance is installed (using the `/ig` parameter), it is not possible to install or maintain the default instance without (a) first uninstalling the non-default instance or (b) specifying the GUID of the default instance through the `/ig` parameter. Therefore, if you support using the `/ig` parameter to install multiple instances, it is recommended that you always use this parameter. This behavior applies to all new InstallShield 2016 projects, as well as projects that are upgraded from earlier versions of InstallShield.

For InstallShield 2008, if a non-multi-instance installation was run on a target system that had multiple instances installed, the instance selection dialog was not displayed; in this case, the default instance was always used. For InstallShield 12 and earlier, if a non-multi-instance installation was run on a target system that had multiple instances installed, the instance selection dialog was displayed; this dialog did not allow a new instance to be displayed (because a non-multi-instance installation does not generate a new random GUID for the new instance). However, the title bar of the dialog indicated that the end user should select an instance to update; it should have indicated that the end user should select an instance to update or maintain.

Changes for SdRegisterUser, SdRegisterUserEx, SdCustomerInformation, and SdCustomerInformationEx in InstallScript MSI Installations

In an InstallScript MSI installation, the following InstallScript text substitutions now map directly to Windows Installer properties:

- IFX_PRODUCT_REGISTEREDOWNER maps to the Windows Installer property USERNAME.
- IFX_PRODUCT_REGISTEREDCOMPANY maps to the Windows Installer property COMPANYNAME.

Therefore, when these InstallScript variables are set and retrieved, the corresponding Windows Installer properties are set and retrieved. The default values for these variables are no longer read from the registry; instead the Windows Installer properties are used by default. To use the previous functionality, you can set the variables manually through the following sample code:

```
// Registered Owner (Only load from registry if pre-loaded value (from log file) is "")
if( !StrLengthChars( IFX_PRODUCT_REGISTEREDOWNER ) ) then
    szValue = "";
    RegDBGetItem( REGDB_WINCURRVER_REGOWNER, szValue );
    if( StrLengthChars( szValue ) ) then
        IFX_PRODUCT_REGISTEREDOWNER = szValue;
    endif;
endif;

// Registered Company (Only load from registry if pre-loaded value (from log file) is "")
if( !StrLengthChars( IFX_PRODUCT_REGISTEREDCOMPANY ) ) then
        szValue = "";
        RegDBGetItem( REGDB_WINCURRVER_REGORGANIZATION, szValue );
        if( StrLengthChars( szValue ) ) then
            IFX_PRODUCT_REGISTEREDCOMPANY = szValue;
        endif;
endif;
```

In an InstallScript MSI installation, the various registration-related dialogs—**SdRegisterUser**, **SdRegisterUserEx**, **SdCustomerInformation**, and **SdCustomerInformationEx**—now set IFX_PRODUCT_REGISTEREDOWNER, IFX_PRODUCT_REGISTEREDCOMPANY, and IFX_PRODUCT_REGISTEREDCOMPANY as appropriate based on the end users' selections. This automatically updates the corresponding Windows Installer properties.

When a null string ("") is specified for either svName or svCompany for the any of the registration-related dialog functions (**SdRegisterUser**, **SdRegisterUserEx**, **SdCustomerInformation**, or **SdCustomerInformationEx**), the functions use the appropriate variable (determined from the corresponding Windows Installer property) as the default. Previously, the variables were used only if both svName and svCompany were null; for **SdCustomerInformation** and **SdCustomerInformationEx**, the values were read directly from the Windows Installer properties, bypassing the script variables.

If a null string ("") is specified for the svSerial parameter, the default value for the serial number field is now set to IFX_PRODUCT_REGISTEREDSERIALNUM. Previously in this case, the dialog displayed no value.

The following InstallScript variables are new or have been changed:

- **DISABLE_PERUSERBTN**—This existing variable indicates that the per-user radio button should be disabled (or hidden) in cases that it would normally be enabled. This variable is always initialized to FALSE. Previously, this value was initialized to TRUE on Windows 9x systems; otherwise, it was FALSE.
- **DISABLE_ALLUSERBTN**—This existing variable indicates that the all-user radio button should be disabled (or hidden) in cases that it would normally be enabled. This variable is always initialized to FALSE. Previously, this value was initialized to TRUE if the operating system was not Windows 9x and the end user was not an administrator or power user.
- **HIDE_DISABLED_BTNS**—This new variable indicates that the all-user and per-user radio buttons should be hidden instead of being disabled. The default value is TRUE. Note that if this variable is set to TRUE, both radio buttons are hidden if either button is determined to be disabled.

The registration dialogs now automatically disable (or hide) the per-user radio button on Windows 9x regardless of the value of DISABLE_ALLUSERBTN. Previously, the per-user radio button was disabled only if DISABLE_PERUSERBTN was FALSE.

Note also that the registration dialogs automatically disable (or hide) the all-users radio button if the end user is not an administrator or power user, regardless of the value of DISABLE_ALLUSERBTN. This behavior has not changed.

These changes apply to all new InstallScript MSI projects that are created in InstallShield 2016, as well as InstallScript MSI projects that were created in earlier versions of InstallShield and then upgraded to InstallShield 2016.

Proxy Server Support

You may want to configure your installation to download certain files only if they are needed on the target system. For example, the Windows Installer engine, the .NET Framework, and some InstallShield prerequisites may already be present on some or most target systems. Instead of embedding these files in your installation (which would increase your overall installation size), you can configure your project so that only the ones that are needed are downloaded at run time.

If your end users access the Internet through a proxy server and your installation is configured to download files, the installation now uses the system proxy settings that are manually configured in Internet Explorer during the download. This occurs even if another browser on the target system is the default browser.

Note that InstallShield does not include support for the Automatically Detect Settings functionality in Internet Explorer. (If end users have the Automatically Detect Settings check box selected in Internet Explorer for their LAN connection and the installation needs to download files, the installation fails because the files cannot be downloaded. If it is possible that your end users may have the Automatically Detect Settings check box selected in Internet Explorer for their LAN connections, you may want to embed all of the files in your installation rather than configure them to be downloaded; if the files are embedded, the failures can be avoided.) However, InstallShield does support the Automatic Configuration Script functionality that is set up for LAN connections in Internet Explorer.

This is the behavior for all new projects in InstallShield 2016, as well as all projects that are created in earlier versions and then upgraded to InstallShield 2016.

In InstallShield 2008 and earlier, the installation attempted to use the proxy server settings that were configured in whatever browser was the default browser. However, this was not always possible, and it caused some problems:

- If Netscape 6 or 7 was the default browser, the Netscape 4 settings were used. If Netscape 8 or 9 was the default browser, the system (Internet Explorer) settings were used.
- If Netscape 4 settings were used, only the proxy server list was read and imported correctly. The proxy bypass list was read, but it was not imported correctly.

- Non–Internet Explorer 4 compatible settings such as the auto-proxy script setting were not imported.
- The method that the installation used for determining the default browser was not compatible on Windows Vista. Therefore, on Windows Vista systems, an installation may not have detected the default browser correctly.

Web Downloader Option to Wrap a 1.x .msi Package into a .cab File Is No Longer Available

InstallShield no longer includes an option to specify that an .msi package should be wrapped into a .cab file. This option was previously available for the Web Downloader type of media in Basic MSI and InstallScript MSI projects in which the Windows Installer 1.1 or 1.2 was included. The recommended method is to use Windows Installer 2.0 or later and to digitally sign the package.

Status Text in InstallScript MSI Projects

The OnFirstUIBefore, OnMaintUIBefore, OnAdminInstallUIBefore, OnAdminPatchUIBefore, OnPatchUIBefore, OnUninstall, and OnResumeUIBefore event handlers in an InstallScript MSI project now include **SetStatusExStaticText** calls by default in all new InstallScript MSI projects. This new default code sets status text for the STATUSEX dialog; the status text that is displayed is now appropriate for the type of operation (first-time installation, maintenance, uninstallation, repair, etc.) being performed. Previously, it was necessary to manually add the **SetStatusExStaticText** calls; otherwise, the status text that was displayed did not correspond with the type of operation being performed.

Note that if you upgrade an InstallScript MSI project from InstallShield 2008 or earlier to InstallShield 2016 and you had overridden one or more of the updated events in the earlier version of your project, you must manually add the **SetStatusExStaticText** code to these events.

Changes for the InstallScript Variable SHELL_OBJECT_FOLDER

You can specify **SHELL_OBJECT_FOLDER** (for InstallScript or InstallScript MSI projects) or **<SHELL_OBJECT_FOLDER>** (for InstallScript projects) in the Display Name setting for a folder in the Shortcuts view. Then you can define the display name for the folder at run time by setting the SHELL_OBJECT_FOLDER variable in your script before the shortcut is created. The shortcut is typically created during file transfer.

To use this functionality in an InstallScript MSI installation, any letters that are specified for the Key Name setting of the folder in the Shortcuts view must be all uppercase (for example, NEWFOLDER1).

SHELL_OBJECT_FOLDER is now initialized to the same value as IFX_PRODUCT_NAME during initialization. Note that these variables are not synchronized once initialized; therefore, if you change one and want the other to change, you must change both manually.

Note that also as part of this change, the default OnFirstUIBefore event handler code in InstallScript MSI projects no longer includes the following line of code:

```
SHELL_OBJECT_FOLDER = @PRODUCT_NAME;
```

If you have an InstallScript MSI project that was created in InstallShield 2008 or earlier and you modified the default OnFirstUIBefore code, InstallShield does not remove this line from your code when you upgrade your project to InstallShield 2016.

To learn more, see SHELL_OBJECT_FOLDER.

Change in Behavior for the InstallScript Function FeatureFileEnum

If you use the question mark (?) as a wild-card character for the szQuery parameter of the InstallScript function FeatureFileEnum, the function now uses the question mark as a substitute for exactly one character. If you used the question mark as a wild-card character with **FeatureFileEnum** in a project that you created in an earlier version of InstallShield, you may need to change your InstallScript code when you upgrade it to InstallShield 2016. In earlier versions of InstallShield, the function used the question mark as a substitute for one or zero characters.

For example, if you create a new InstallScript project with default settings and add the following InstallScript code to your project, the dialog that is displayed at run time does not display anything.

```
#include "ifx.h"

   LIST listFiles;

program

   listFiles = ListCreate( STRINGLIST );
   FeatureFileEnum( MEDIA, "DefaultFeature", "Default?Component", listFiles, NO_SUBDIR );
   SdShowInfoList( "", "", listFiles );

endprogram
```

In installations that were created earlier versions of InstallShield, the dialog displayed `DefaultComponent`.

All InstallScript Installations Now Support Showing Update UI

All InstallScript projects now support showing the update user interface (UI). Previously, the General Information view and the Project Settings dialog box had a confusing option that enabled the update UI to be disabled. If the update UI was disabled, the MEDIA_FLAG_UPDATEMODE_SUPPORTED flag was not set for MEDIA_FIELD_MEDIA_FLAGS, and the OnSetUpdateMode event handler did not set the UPDATEMODE variable when appropriate. As a result, the update UI was not shown.

Note that changing the value of ISShowUpdateUI in the InstallShield table (which the previous option updated) no longer has any effect.

It is possible to duplicate the previous functionality through the following methods:

- Override the OnSetUpdateMode event and update the function to return before setting the UPDATEMODE variable. The installation never shows the update UI.
- Override the OnMaintUIBefore event and change the call for FeatureRemoveAllInMediaAndLog to FeatureRemoveAllInMedia.

Changes for the Certified for Windows Vista Validation Suites

The two Certified for Windows Vista validation suites that help you determine whether your installation or merge module meets the requirements for the Windows Vista logo program have been revised. They now consist of only the InstallShield ICEs (ISICEs). The Microsoft-created ICEs (ICE01 through ICE99) have been removed, since they are available in the Full MSI Validation Suite for installation packages and in the Merge Module Validation Suite for merge modules.

In addition, these two suites have been renamed:

- The new name for the Certified for Windows Vista Validation Suite (plus InstallShield ICEs) is *InstallShield Certified for Windows Vista Validation Suite*.

- The new name for the Certified for Windows Vista Merge Module Validation Suite (plus InstallShield ICEs) is *InstallShield Certified for Windows Vista Merge Module Validation Suite*.

For more information about the validation and the Certified for Windows Vista program, see:

- Requirements for the Windows Logo Program
- Validating Projects

Updated InstallScript Objects and Object Templates

A number of improvements were made to the InstallScript objects and InstallScript object templates that are available for use with InstallShield 2016 InstallScript projects.

If you have an Internet connection, you can use the Check for Updates feature in InstallShield to obtain the updated InstallScript objects and object templates for the version of InstallShield that you are using. To check for updates: On the Tools menu, click Check for Updates. InstallShield launches FlexNet Connect, which checks for updates.

Changes that Affect New Projects but Not Upgraded Projects

This section describes changes to InstallShield that may affect new projects but not projects that are upgraded from earlier versions. Note that you may need to make manual changes to upgraded projects.

Unicode and ANSI Versions of the Setup.exe and Update.exe Bootstrappers

InstallShield now enables you to specify whether you want to create a Unicode version or an ANSI version of the **Setup.exe** setup launcher for a Basic MSI project. Previously, if your Basic MSI project included a setup launcher, InstallShield always built an ANSI version; it did not include support for building a Unicode version.

A Unicode setup launcher can correctly display double-byte characters in the user interface of the setup launcher, regardless of whether the target system is running the appropriate code page for the double-byte-character language. An ANSI setup launcher displays double-byte characters in the setup launcher dialogs if the target system is running the appropriate code page. However, it displays garbled characters instead of double-byte characters in those dialogs if the target system is not running the appropriate code page.

If you create a new release in a Basic MSI project in InstallShield 2016, the default setup launcher type is Unicode. In addition, if you create a new patch configuration or a new QuickPatch project in InstallShield 2016, the default update launcher type is Unicode.

If you upgrade a Basic MSI project from InstallShield 2008 or earlier to InstallShield 2016, the setup launcher type for any existing releases is ANSI. You can override the type if appropriate.

Similarly, if you upgrade a Basic MSI project or a QuickPatch project from InstallShield 2008 or earlier to InstallShield 2016, the update launcher type for any existing patch is ANSI. You can override the type if appropriate.

Dynamic File Links

When you add or modify a dynamic file link in a project, you can specify which component creation method you want InstallShield to use: a new best practice method or the previously available one-component-per-directory method. These methods are applicable to dynamic file linking in Basic MSI, InstallScript MSI, and Merge Module projects.

If you create a new dynamic file link in InstallShield 2016, InstallShield uses the best practice method by default.

All dynamic file links that are created in InstallShield 2008 or earlier use the one-component-per-directory method. If you have a project with dynamic file links and you upgrade it from InstallShield 2008 or earlier to InstallShield 2016, InstallShield continues to use the one-component-per-directory method for creating the components of those already present dynamic file links. For any new dynamic file links that you create in the upgraded project, the best practice method is used by default. For detailed information about the two component creation methods, as well as guidance on which method you should use, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.

IIS Web Sites and Associated Components

InstallShield now includes support for installing IIS Web sites without any virtual directories. In addition, InstallShield now lets you associate a Web site with a component. As a result of these enhancements, a Web site is created on a target machine if a virtual directory or a component that is associated with it is installed.

If you add a new Web site through the Internet Information Services view in InstallShield 2016, InstallShield automatically associates that Web site with a component. If you upgrade an InstallShield 2008 or earlier project that already has an IIS Web site, InstallShield does not automatically associate that Web site with a component. Use the General tab that InstallShield displays when you select a Web site in the Internet Information Services view if you want to associate the selected Web site with a component.

Note that if a Web site is associated with a component, the Web site's **Delete Web Site on Uninstall** check box now corresponds with the Permanent setting for that component in a Basic MSI or InstallScript MSI project, or with the Uninstall setting for that component in an InstallScript project. That is, if you select or clear the **Delete Web Site on Uninstall** check box for a Web site, InstallShield automatically updates the value of the component's Permanent setting or Uninstall setting, as appropriate.

Simplification of QuickPatch Packages

The new Streamline QuickPatch setting on the Advanced tab in a QuickPatch project determines how InstallShield builds QuickPatch packages. A streamlined QuickPatch package typically has fewer new subfeatures and custom actions than a non-streamlined QuickPatch package.

In some cases, InstallShield cannot streamline the QuickPatch package. For example, if you configure the QuickPatch package to remove an installed file, InstallShield cannot streamline it.

When you create a new QuickPatch project, the default value for the Streamline QuickPatch setting is Yes. However, when you upgrade a QuickPatch project from InstallShield 2008 or earlier to InstallShield 2016, the value for this setting is No. You can change this value if appropriate. For more information, see Specifying Whether to Streamline the QuickPatch Package.

Default Condition for the SetARPINSTALLLOCATION Custom Action

By default, all new Basic MSI and InstallScript MSI projects contain the built-in InstallShield custom action SetARPINSTALLLOCATION. This custom action, which sets the value of the ARPINSTALLLOCATION property to the fully qualified path for the product's primary folder, is scheduled for the Installation Execute sequence, and it has no condition. In InstallShield 2008 and earlier, the default condition for this custom action was Not Installed. With this default Not Installed condition, the custom action is not run during maintenance mode, and this results in a blank value for the ARPINSTALLLOCATION property. To avoid this behavior in a project that you upgrade from InstallShield 2008 or earlier to InstallShield 2016, open the Custom Actions and Sequences view and delete the Not Installed value from the Install Exec Condition setting for this custom action.

New Sequence Location for the MigrateFeatureStates Action

By default, the MigrateFeatureStates action is now sequenced immediately after the CostFinalize action in all new Basic MSI and InstallScript MSI projects. If you use the SQL Scripts view to add SQL support to the new project, InstallShield now schedules the InstallShield built-in custom action ISSQLServerFilteredList after the MigrateFeatureStates action. In InstallShield 2008 and earlier, the ISSQLServerFilteredList was sequenced before the MigrateFeatureStates action, and this caused run-time error 2601. To avoid this error in a project that you upgrade from InstallShield 2008 or earlier to InstallShield 2016, open the Custom Actions and Sequences view and move the ISSQLServerFilteredList action after the MigrateFeatureStates in the Installation User Interface sequence.

Template Summary Property Changes

You can specify a value for the Template Summary property in the General Information view or for a product configuration in the Releases view. If you try to enter an invalid value (for example, if you indicate multiple platforms or you use more than one semicolon), InstallShield displays a warning message box and does not allow you to change the value for the property.

Earlier versions of InstallShield permitted you to enter invalid values. If you upgrade an InstallShield 2008 or earlier project to InstallShield 2016, InstallShield does not automatically change the invalid value. However, you can manually update the value if appropriate. For more information, see Using the Template Summary Property.

Welcome Assistant Removed

The Welcome Assistant, which was displayed the first time that you opened InstallShield, is no longer included.

InstallShield MSIPackageDiff Replaced by InstallShield MSI Diff

The tool called InstallShield MSIPackageDiff is no longer included with InstallShield. It has been replaced by a more powerful tool called InstallShield MSI Diff. You can launch InstallShield MSI Diff from within InstallShield: On the Tools menu, point to Difference, and then click InstallShield MSI Diff.

For more information, see InstallShield MSI Diff.

Upgrading Projects from InstallShield 12 or Earlier

The following information describes changes that may affect projects that are upgraded from InstallShield 12 or earlier to InstallShield 2016.

Upgrading Projects Created in Earlier Versions of InstallShield

If you use InstallShield 2016 to open a project that was created with an earlier version, InstallShield 2016 displays a message box that asks you if you want to convert the project to the new version. If you reply that you do want to convert it, InstallShield creates a backup copy of the project with a file extension such as .765 before converting it. Delete the .765 part from the original project's file name if you want to reopen the project in the earlier version of InstallShield. Note that you cannot open InstallShield 2016 projects in earlier versions of InstallShield.

You can upgrade projects that were created with the following versions of InstallShield to InstallShield 2008: InstallShield 12 and earlier, InstallShield DevStudio, InstallShield Professional 7 and earlier, and InstallShield Developer 8 and earlier. Note that projects that were created with InstallShield MultiPlatform or InstallShield Universal cannot be upgraded to InstallShield 2008.

End of Support for Windows 9x, Windows NT 4, and Windows Me on Target Systems

InstallShield no longer supports the creation of installations for Windows 9x, Windows NT 4, and Windows Me systems. If end users have one of these operating systems on their computer and they try to run an installation that was built with InstallShield 2016, unexpected results may occur, unless the project includes launch conditions that prevent end users from running the installation on any of these legacy operating systems.

For Basic MSI and InstallScript MSI projects, you may want to consider adding launch conditions that display a message if end users are running your installation on any of the legacy systems. For InstallScript projects, you may want to consider adding InstallScript code that uses the SYSINFO structure variable to check for these legacy systems, and then display a message if any of these legacy systems are present.

To learn more, see:

- Specifying Operating System Requirements in the Project Assistant
- Platforms Tab
- Platforms Dialog Box
- Modify Property Dialog Box/Release Wizard—Platforms Panel

COM Extraction

When you use InstallShield to extract COM information from a COM server, InstallShield puts the data in the Registry table, instead of in the **TypeLib** table. Microsoft strongly advises against using the TypeLib table.

Unused Directories Automatically Removed from .msi File at Build Time by Default

Note that if you upgrade a Basic MSI, InstallScript MSI, or Merge Module project that was created in InstallShield 12 or earlier to InstallShield 2016, the new Keep Unused Directories setting on the Build tab in the Releases view is set to No. Therefore, if a directory that is listed in the Directory column of the Directory table is not referenced in any known location in the .msi file, InstallShield removes it from the Directory table of the .msi file that it creates at build time. For Basic MSI and InstallScript MSI projects, this occurs after any merge modules are merged, but only directories that are present in the .msi file are removed; therefore, if a merge module contains new unused directories in its Directory table, the new unused directories are added to the installation.

Changes for ALLUSERS and for the CustomerInformation Dialog

Beginning with InstallShield 2008, the **ALLUSERS** property is set to 1 by default in all new Basic MSI projects. This is the recommended implementation, since most installations must be run in a per-machine context with administrative privileges.

If you upgrade a project that was created with InstallShield 12 or earlier to InstallShield 2016, InstallShield does not automatically change the value of the **ALLUSERS** property or add this property if it was not defined in the earlier project.

Also new with InstallShield 2008, by default, the CustomerInformation dialog in all new Basic MSI projects does not display the radio button group that enables end users to specify whether they want to install the product for all users or for only the current user. This is the recommended implementation for this dialog.

If you upgrade a project that was created with InstallShield 12 or earlier to InstallShield 2016, InstallShield does not automatically change the CustomerInformation dialog.

To learn more, see:

- Per-User vs. Per-Machine Installations
- ALLUSERS

Windows Server 2003 Conditions and 64-Bit Windows XP Conditions for Setup Prerequisites

The operating system version number is 5.2 for both Windows Server 2003 and 64-bit Windows XP. As a result, prerequisites that were created in InstallShield 12 and that required Windows Server 2003 could be installed on 64-bit Windows XP systems, and those that required Windows XP could not be installed on 64-bit Windows XP systems.

To resolve this issue, the Setup Prerequisite Editor in InstallShield 2008 has been enhanced to enable you to specify whether the target system is required to be a workstation, a server, or a domain controller.

To resolve this issue for an existing prerequisite that includes a Windows Server 2003 requirement or a 64-bit Windows XP requirement, open the prerequisite in the Setup Prerequisite Editor in InstallShield 2008. On the Conditions tab, select the condition that needs to be corrected and click Modify. In the **Select which platform the prerequisite should be run on** box, select the appropriate operating system requirement. Doing this correctly sets the new Product (OS) Type setting to the appropriate workstation, server, or domain controller value.

Automation Interface

The Display Save Options dialog setting was removed from the Releases view. Therefore, the WebSaveOptionsDlg property, which corresponds with that setting, is no longer available for the ISWiRelease object of automation interface.

New Default Value for the Cache Path Setting for a Release

The default value for the Cache Path setting for a compressed release in the Releases view is now set to **[LocalAppDataFolder]Downloaded Installations**. The previous default value was **[WindowsFolder]Downloaded Installations**, which may not be available to users on locked-down systems. If you migrate a project from InstallShield 12 or earlier to InstallShield 2016, the Cache Path setting is not automatically changed. Therefore, you may want to change that value.

InstallScript One-Click Install Installations

Setup.exe is no longer used as the setup player for InstallScript One-Click Install installations; **Setup.ocx** is now used instead. In order for a **Setup.ocx** file to be included in an installation, the new Generate One-Click Install setting in the Releases view must be set to Yes. If you upgrade an InstallScript project from InstallShield 12 or earlier to InstallShield 2016 and the Create Default Web Page setting in the Releases view is set to Yes, InstallShield sets the Generate One-Click Install setting to Yes automatically during the upgrade. However, if the Create Default Web Page setting is set to No and you intend to distribute the installation over the Internet, you must manually select Yes for the Generate One-Click Install setting after upgrading the project.

All functionality that is related to the Save And/or Run Setup dialog box has been removed from InstallShield. If you want to give end users the option of downloading and saving the installation (or creating an icon on the desktop), you need to implement this within your installation's Web page, and handle the download and save operations through the Web page.

To pass data to a launched One-Click Install installation, use the is::CmdLine parameter and the CMDLINE script variable. Previously, it was possible to pass data from the Web page to the installation; however, this support has been removed. Note that the only valid value for the first parameter of SetProperty is now is::CmdLine.

To learn more about One-Click Install installations, see One-Click Install Installations in InstallScript Projects.

Patch Creation for Basic MSI and InstallScript MSI Projects

InstallShield now uses version 3.1 of Patchwiz.dll to create patches.

DemoShield Support

DemoShield is no longer being sold. In addition, it is no longer supported. Therefore, InstallShield no longer includes any DemoShield integration.

Changes for the Windows Vista Validation Suites

The validation suites that help you determine whether your installation meets installation requirements for the Windows Vista logo program have been renamed:

- The new name for the Windows Vista Quality Validation Suite (plus InstallShield ICEs) is *Certified for Windows Vista Validation Suite (plus InstallShield ICEs)*.
- The new name for the Windows Vista Quality Merge Module Validation Suite (plus InstallShield ICEs) is *Certified for Windows Vista Merge Module Validation Suite (plus InstallShield ICEs)*.

In addition, three of the InstallShield ICEs have been moved to the InstallShield Best Practice Suite.

For more information about the validation and the Certified for Windows Vista program, see:

- Requirements for the Windows Logo Program
- Validating Projects

To learn about the validators that are now available as InstallShield Best Practices, see:

- ISBP17 (which was previously known as ISICE13)
- ISBP18 (which was previously known as ISICE14)
- ISBP19 (which was previously known as ISICE15)

Upgrading Projects from InstallShield 11.5 or Earlier

For InstallShield 12, a number of improvements were made to the InstallScript MSI architecture to resolve issues with security (COM/DCOM) and other areas that can cause some installations to fail for various reasons. The architecture was improved for InstallShield 12 to resolve these issues and to make InstallScript MSI a more reliable project type. The improvements also help increase the reliability of InstallScript projects, as well as Basic MSI projects that include InstallScript custom actions.

The following table compares the earlier design with the current design:

Table 2-9 • Architecture Changes

In InstallShield 11.5 and Earlier	In InstallShield 12 and Later
Each InstallScript custom action is handled by **ISScriptBridge.dll** (a C++ MSI DLL), which forwards actual script execution to the ongoing **IDriver.exe** process.	Each InstallScript custom action is handled by **ISSetup.dll** (a single C++ MSI DLL), which contains the full InstallScript engine.

Table 2-9 • Architecture Changes (cont.)

In InstallShield 11.5 and Earlier	In InstallShield 12 and Later
A set of engine files is required. They must be installed by **ISScript.msi** before the primary installation begins.	No engine files or **ISScript.msi** file is required. **ISSetup.dll** is self-contained.
All InstallScript custom actions execute in a shared **IDriver.exe** process. The **IDriver.exe** process remains resident until a final shutdown custom action is required.	Each InstallScript custom action is independent and has its own self-contained lifecycle. From the custom action entry point, **ISSetup.dll** starts the script engine, executes the relevant script, and shuts down the engine.

Advantages of the Architecture Changes That Were Introduced in InstallShield 12

The architecture in InstallShield 12 and later has several advantages over the earlier architecture:

- End users no longer encounter error 1607, error 1608, or other COM/DCOM run-time errors that are related to finding the running **IDriver.exe** file. When these errors occurred under the earlier model, they were difficult to resolve, often requiring changes to DCOM settings. Also, the reliance on the running object table made the model brittle across the spectrum of usage scenarios, including Fast User Switching and Windows Terminal Services.
- The engine binding is static. No shared engine files are installed to **Program Files\Common Files\InstallShield**. The reliance in InstallShield 11.5 and earlier on shared engine files made the model brittle because separate installations were not isolated. Changes to one engine version could break an existing installation. Isolation, especially for an installer, is one key to reliability.
- The **ISScript.msi** file (the InstallScript engine installer) is no longer needed. The earlier model required an **ISScript.msi** file to execute prior to the primary .msi file. Because the full application was not contained in a single .msi file, the installation essentially required a bootstrap for Basic MSI installations that had InstallScript custom actions. This was very inflexible for enterprises that use Active Directory for managed environments. Also, it was confusing for advanced users who expect the main .msi file to be self-contained. InstallScript MSI installations still require a bootstrap, so that requirement is no different than the earlier architecture; however, Basic MSI installations with InstallScript custom actions no longer require a bootstrap.
- Fewer InstallShield custom actions are needed to run the installation. The earlier model relied on a deep coupling between startup and shutdown custom actions that could easily fail. If one of the required custom actions was deleted or moved, the installation would not work properly.

Disadvantages of the Architecture Changes That Were Introduced in InstallShield 12

The disadvantages of the architecture in InstallShield 12 and later include the following:

- InstallScript events are not available in Basic MSI and merge module projects; therefore, all InstallScript code for these project types must be run by InstallScript custom actions. Global variables do not share state between these custom action invocations. Therefore, you can declare a global variable in one InstallScript custom action script and then use that global variable throughout the script. However, if your Basic MSI installation has a second InstallScript custom action, the global variable declared in the first script is not available to the second script.

With the earlier model, InstallScript developers were able to use script code in any of the InstallScript events in Basic MSI projects, and any InstallScript global variables shared state. This allowed setup authors to write custom actions with a more holistic view and sense of continuity across the installation. However, because global variables are generally discouraged in programming, the new limitation should actually result in better-written InstallScript code.

- In-memory objects must be serialized in order to be shared between custom action invocations. With the earlier model, InstallScript developers could store complex object-based data in global variables. As with the aforementioned disadvantage, this may actually result in better InstallScript code.

Upgrading InstallShield 11.5 and Earlier Projects

The extensive rearchitecture that has been introduced in InstallShield 12 may require some manual changes when you upgrade projects that were created with InstallShield 11.5 or earlier to InstallShield 2016.



Task ***To upgrade your project:***

1. Before you convert your project, create a backup version of your project file and any InstallScript files.
2. Open InstallShield 2016.
3. On the **File** menu, click **Open**. The **Open** dialog box opens.
4. Browse to select the project file (.ism) of the InstallShield 11.5 or earlier project that you want to upgrade. A dialog box opens, prompting you to specify whether you want to upgrade the project.
5. Click Yes.

InstallShield upgrades your project and saves a backup copy of the project file (.ism).

To learn about possible manual changes that you may need to make to upgraded projects, see the following:

- Upgrading InstallShield 11.5 or Earlier Basic MSI Projects that Have InstallScript Custom Actions
- Upgrading InstallShield 11.5 or Earlier InstallScript MSI Projects
- Upgrading InstallShield 11.5 or Earlier InstallScript Projects
- Upgrading InstallShield 11.5 or Earlier QuickPatch Projects that Have InstallScript Custom Actions
- Creating Standard Patches for InstallShield 11.5 and Earlier InstallScript MSI Projects
- Upgrading InstallShield 11.5 or Earlier InstallScript MSI Object Projects or Projects that Contain This Type of Object

Upgrading InstallShield 11.5 or Earlier Basic MSI Projects that Have InstallScript Custom Actions

The following sections explain details about upgrading Basic MSI projects that were created in InstallShield 11.5 or earlier to InstallShield 2016.

Global Variables, Global Pointers, and SUPPORTDIR

With InstallShield 12 and later, when a Basic MSI installation executes an InstallScript custom action, the compiled InstallScript is loaded before the action is called, and it is unloaded after the action completes. Thus, each InstallScript custom action executes in its own session with complete InstallScript engine loading and unloading. This behavior is different than with InstallShield 11.5 and earlier: the compiled InstallScript was loaded once before the first InstallScript custom action that was used by the InstallScript was executed, and it was unloaded at the end of the installation after all InstallScript custom actions were completed.

A major implication of this change in behavior is that global variables and pointers are no longer maintained between individual InstallScript custom action calls:

- If you need to store a value across multiple custom action calls, you must use some external mechanism such as the registry, Windows Installer properties, or an external data file to store the information between calls. If you choose to use Windows Installer properties in deferred, commit, or rollback InstallScript custom actions, see the guidelines in Windows Installer Properties and Deferred, Commit, and Rollback InstallScript Custom Actions.
- If you need to use a COM object or some other global object across custom action calls, you must initialize the object for each individual custom action call in order for the object to be valid.

Another implication of this change is that each custom action initializes and uses its own individual **SUPPORTDIR**. Therefore, you cannot share information across individual calls using files in **SUPPORTDIR**, since each custom action invocation will have its own unique **SUPPORTDIR**. You can share information using **FOLDER_TEMP** or some other file location.

Note that **FOLDER_TEMP** may not be the same path for all of your InstallScript custom actions. If you have some InstallScript custom actions that run in system context and some that do not, they will have different temp paths if the package is running in an elevated state. The InstallScript custom actions run in the context of a different user, so storing files in the temp directory and retrieving it later may not work in certain scenarios.

Windows Installer Properties and Deferred, Commit, and Rollback InstallScript Custom Actions

Deferred, commit, and rollback InstallScript custom actions in Basic MSI installations have access to only some of the built-in Windows Installer properties: **CustomActionData**, **ProductCode**, and **UserSID**. If you want an InstallScript custom action to access any other properties (such as **SUPPORTDIR**) during deferred, commit, or rollback execution, you need to pass them as **CustomActionData**. You can do so by scheduling an immediate set-a-property type of custom action (or, for example, an immediate InstallScript custom action with the **MsiSetProperty** function) to set a property that matches the name of the custom action. The value of this property is then available in the **CustomActionData** property within the deferred custom action.

For example, if you want to access a property such as **SUPPORTDIR**, you could create an immediate custom action that is called MyCustomActionName and that sets the MyCustomActionName property to [SUPPORTDIR], and then substitute "SUPPORTDIR" with "CutomActionData" in the MsiGetProperty call:

```
MsiGetProperty(hMSI, "CustomActionData", szSupportDir, nLen)
```

For more details, see the following:

- Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions
- Obtaining Context Information for Deferred Execution Custom Actions

Note that failure to compensate for an unavailable Windows Installer property may cause unexpected results during installation. For example, since in a deferred action, `MsiGetProperty(hMSI, "INSTALLDIR", szInstallDir, nLen)` sets szInstallDir to the empty string, szInstallDir ^ szFile may refer to a file in the current directory instead of a file in [INSTALLDIR]. The current directory for a deferred custom action is often [SystemFolder].

Deferred, commit, and rollback InstallScript custom actions do not have access to the **ProductLanguage** property. If your installation includes multilanguage support and it also has a deferred, commit, or rollback InstallScript custom action that needs access to the language in which the end user is running the installation, the installation assumes that this language is the installation's default language. This could be a problem if an end user runs the installation in a language other than the default language. However, deferred, commit, and rollback custom actions do not typically display any user interface, so this would usually not be a problem.

Predefined InstallScript Event Handler Functions

The predefined InstallScript event handler functions are no longer available in Basic MSI projects with InstallScript custom actions. In InstallShield 11.5 and earlier, the following InstallScript event handler functions were available for Basic MSI projects:

- OnBegin
- OnMoving
- OnMoved
- OnEnd

InstallShield no longer supports these event handler functions for Basic MSI projects that have InstallScript custom actions. They are not called once the project is rebuilt with InstallShield 12 and later. Note that these event handler functions still compile in InstallShield 12 and later; they are just not called.

If you use these event handler functions in your project, you need to manually schedule custom actions that call these functions. To learn how, see Creating and Scheduling InstallScript Custom Actions that Call InstallScript Event Handlers for Basic MSI Projects.

InstallShield InstallScript Scripting Engine Merge Module

Beginning with InstallShield 12, the InstallShield scripting engine merge module is no longer available for inclusion in Basic MSI projects. In InstallShield 11.5 and earlier, you could use this merge module to distribute the InstallScript engine for installations that did not include **Setup.exe**. InstallShield 12 and later automatically includes the InstallScript engine as part of **ISSetup.dll** in Basic MSI installations that have InstallScript custom actions, regardless of whether **Setup.exe** is used. Therefore, you do not need this merge module in InstallShield 2016 projects.

If you use InstallShield 2016 to try to build a release for a project that has this merge module, you may encounter a build error such as the following one:

```
ISDEV : error -4075: File not found. An error occurred merging Module
    'InstallShieldScriptingEngine.4F635B62_07BF_4779_B74E_D80C29D508E3:0' for Feature 'NewFeature1'.
```

To resolve this build error, remove this merge module from your project; you can do so by clearing its check box in the Redistributables view.

Changes to the InstallScript Engine Files

With InstallShield 12 and later, Basic MSI installations that include InstallScript custom actions no longer install InstallScript engine files to the following directory:

<COMMONFILES>\InstallShield\Driver\<Version>\Intel 32

For InstallShield 11.5 and earlier Basic MSI projects with InstallScript custom actions, several files were stored in the **Binary** table:

- **Setup.inx**
- **Isconfig.ini**
- **Isrt.dll**
- **ISScriptBridge.dll**
- **_isresXXXX.dll** (where XXXX is the language—one .dll was included for each language included in the installation)
- **StringxXXXX.txt** (where XXX is the language—one .txt was included for each language in the installation)

For InstallShield 12 and later, all of these files (except **ISScriptBridge.dll**, which is no longer used) are stored inside the **ISSetup.dll** file, and that is the only file that is stored in the **Binary** table.

The InstallScript engine file changes have not been known to cause any upgrade issues. These changes are reported for informational purposes.

Creating and Scheduling InstallScript Custom Actions that Call InstallScript Event Handlers for Basic MSI Projects

For InstallShield 12 and later, the predefined InstallScript event handler functions are no longer available in Basic MSI projects with InstallScript custom actions. In InstallShield 11.5 and earlier, the following InstallScript event handler functions were available for Basic MSI projects and InstallScript MSI object projects:

- OnBegin
- OnMoving
- OnMoved
- OnEnd

InstallShield no longer supports these event handler functions for Basic MSI projects that have InstallScript custom actions. They are not called once the project is rebuilt with InstallShield 2016. Note that these event handler functions still compile in InstallShield 2016; they are just not called.

Similarly, these same predefined InstallScript event handler functions are not supported in InstallScript MSI object projects that are converted to merge module projects automatically when they are opened in InstallShield 2016. The functions are not called once the converted merge module project is built in InstallShield 2016.

If you have these events in your Basic MSI or InstallScript MSI object project and you upgrade your project to InstallShield 2016, you need to manually schedule custom actions that call the event handler functions. The following instructions explain how to do so.



Task ***To manually schedule InstallScript custom actions that call the predefined InstallScript event handler functions:***

1. Open the upgraded project in InstallShield 2016.
2. Make the appropriate changes to your InstallScript file:
   a. In the View List under **Behavior and Logic**, click **InstallScript**.
   b. Find the OnBegin, OnMoving, OnMoved, and OnEnd event handler functions in your script. You can quickly find a function by clicking the function name in the center pane of the view.
   c. Rename the functions to an alternate name to avoid conflicts with existing function prototypes automatically included in **ifx.h**. For example:
      - MyOnBegin
      - MyOnMoving
      - MyOnMoved
      - MyOnEnd
   d. Update the existing functions to take a single HWND parameter. For example:

      ```
      function MyOnBegin(hMSI) begin end;
      ```

   e. Add appropriate prototypes for these new functions:

      ```
      export prototype MyOnBegin(HWND);
      export prototype MyOnEnd(HWND);
      export prototype MyOnMoved(HWND);
      export prototype MyOnMoving(HWND);
      ```

3. Add InstallScript custom actions that call your renamed InstallScript event handler functions:
   a. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI projects) or **Custom Actions** (in merge module projects).
   b. In the center pane, right-click the **Custom Actions** explorer and then click **New InstallScript**. InstallShield adds an InstallScript custom action.
   c. Type a name for the custom action; use the same name that you used to rename the InstallScript functions. For example:
      - MyOnBegin
      - MyOnMoving
      - MyOnMoved
      - MyOnEnd
   d. Select the new InstallScript custom action that you created.
   e. Set the **Function Name** setting to the name of the InstallScript function.
   f. For the **In-Script Execution** setting, specify the value that is indicated in the table below.
4. Schedule the InstallScript custom action for the appropriate part of the installation:

a. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.

b. Right-click the action or dialog that you want your InstallScript custom action to follow and click **Insert**. The **Insert Action** dialog box opens, providing a list of all the actions and dialogs that are currently associated with your project.

c. Select the InstallScript custom action that you created. If appropriate, enter a condition in the **Condition** box for launching the InstallScript event.

d. Click **OK**.

To match the previous functionality as closely as possible, set the in-script execution in step 3f and schedule the InstallScript custom actions in step 4b as follows:

Table 2-10 • Scheduling the Custom Actions

Custom Action	In-Script Execution	Location in the Sequence
MyOnBegin	Immediate (default)	One of the first actions in the Installation User Interface sequence.
MyOnMoving	Deferred in system context	In the Installation Execution sequence, between the InstallInitialize and AllocateRegistrySpace actions.
MyOnMoved	Deferred in system context	In the Installation Execute sequence, between the ScheduleReboot and InstallFinalize actions.
MyOnEnd	Immediate (default)	The last event in the Installation Execute sequence after the InstallFinalize action. (In previous releases, OnEnd was called after all other events as a result of the installation completing.)

Note the following:

- Global variables and pointers are no longer maintained between individual InstallScript custom action calls. In addition, each InstallScript custom action initializes and uses its own individual **SUPPORTDIR**. Therefore, you cannot share information across individual calls using files in **SUPPORTDIR**. For more information, see Global Variables, Global Pointers, and SUPPORTDIR.
- Most Windows Installer properties are not available to deferred, commit, and rollback InstallScript custom actions. For more information, see Windows Installer Properties and Deferred, Commit, and Rollback InstallScript Custom Actions.

Upgrading InstallShield 11.5 or Earlier InstallScript MSI Projects

The following sections explain details about upgrading InstallScript MSI projects that were created in InstallShield 11.5 or earlier to InstallShield 2016.

Global Variables, Global Pointers, and SUPPORTDIR

With InstallShield 2016 and earlier versions, when an InstallScript MSI installation initializes, it loads the compiled InstallScript file, and it calls InstallScript events in this “main” loaded InstallScript file as necessary during the installation.

However, for InstallShield 12 and later, when a function within the InstallScript is called by the Windows Installer engine as an InstallScript custom action, the custom action behaves like an InstallScript custom action in a Basic MSI installation. That is, the function is executed in a completely separately loaded script instance; the script is loaded before the action is called and unloaded after the action completes. Thus, each InstallScript function that is called by the Windows Installer engine as an InstallScript custom action executes in its own session with complete InstallScript engine loading and unloading. This behavior is different than with InstallShield 11.5 and earlier: the custom actions were called in the single main loaded script file instance that was loaded during initialization and unloaded at the end of the installation.

A major implication of this change in behavior is that InstallScript functions that are called as InstallScript custom actions no longer have access to global variables and pointers that are maintained by the main executing script file:

- If you need to interact with the main executing InstallScript file to perform tasks such as using or changing global variables or modifying objects that are maintained by the main executing script file (which includes updating the status bar or logging script operations for uninstallation), your function must be called from an InstallScript event—not as an InstallScript custom action. Functions that are called from InstallScript events are executed in the main InstallScript instance. (Exception: Some InstallScript error events such as OnFilesInUse are called as a result of an error message from the Windows Installer engine. The Windows Installer engine launches these InstallScript error events as InstallScript custom actions; thus, the limitations of InstallScript custom actions also apply to these InstallScript events.)
- If you need to store a value across multiple InstallScript custom action calls, you must use some external mechanism such as the registry, Windows Installer properties, or an external data file to store the information between calls. If you choose to use Windows Installer properties in deferred, commit, or rollback InstallScript custom actions, see the guidelines in Windows Installer Properties and Deferred, Commit, and Rollback InstallScript Custom Actions.
- If you need to use a COM object or some other global object across custom action calls, you must initialize the object for each individual custom action call in order for the object to be valid.

Another implication of this change is that each custom action initializes and uses its own individual **SUPPORTDIR**. Therefore, you cannot share information across individual calls using files in **SUPPORTDIR**, since each custom action invocation will have its own unique **SUPPORTDIR**. You can share information using **FOLDER_TEMP** or some other file location.

Note that **FOLDER_TEMP** may not be the same path for all of your InstallScript custom actions. If you have some InstallScript custom actions that run in system context and some that do not, they will have different temp paths if the package is running in an elevated state. The InstallScript custom actions run in the context of a different user, so storing files in the temp directory and retrieving it later may not work in certain scenarios.

Windows Installer Properties and Deferred, Commit, and Rollback InstallScript Custom Actions

Deferred, commit, and rollback InstallScript custom actions in InstallScript MSI installations have access to only some of the built-in Windows Installer properties: **CustomActionData**, **ProductCode**, and **UserSID**. If you want an InstallScript custom action to access any other properties (such as **SUPPORTDIR**) during deferred, commit, or rollback execution, you need to pass them as **CustomActionData**. You can do so by scheduling an immediate set-a-property type of custom action (or, for example, an immediate InstallScript custom action with the MsiSetProperty function) to set a property that matches the name of the custom action. The value of this property is then available in the **CustomActionData** property within the deferred custom action.

For example, if you want to access a property such as **SUPPORTDIR**, you could create an immediate custom action that is called MyCustomActionName and that sets the MyCustomActionName property to [SUPPORTDIR], and then substitute "SUPPORTDIR" with "CutomActionData" in the MsiGetProperty call:

```
MsiGetProperty(hMSI, "CustomActionData", szSupportDir, nLen)
```

For more details, see the following:

- Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions
- Obtaining Context Information for Deferred Execution Custom Actions

Note that failure to compensate for an unavailable Windows Installer property may cause unexpected results during installation. For example, since in a deferred action, `MsiGetProperty(hMSI, "INSTALLDIR", szInstallDir, nLen)` sets szInstallDir to the empty string, szInstallDir ^ szFile may refer to a file in the current directory instead of a file in [INSTALLDIR]. The current directory for a deferred custom action is often [SystemFolder].

Deferred, commit, and rollback InstallScript custom actions do not have access to the **ProductLanguage** property. If your installation includes multilanguage support and it also has a deferred, commit, or rollback InstallScript custom action that needs access to the language in which the end user is running the installation, the installation assumes that this language is the installation's default language. This could be a problem if an end user runs the installation in a language other than the default language. However, deferred, commit, and rollback custom actions do not typically display any user interface, so this would usually not be a problem.

Changes to the DoInstall Function

The **DoInstall** function now launches the setup executable file of the child installation instead of using the already-running installation engine to run the child installation. This change should be automatic; that is, it should not require any changes to existing InstallScript code. The child installation may take slightly longer to initialize than in previous versions because the installation executable file needs to initialize. For more information, see DoInstall.

Note that now calling **DoInstall** is similar to calling **LaunchAppAndWait**. When the installation is run from any removable media, such as a CD-ROM or a floppy disk, the **Setup.exe** file on Disk1 may not be available during the entire installation. (If **Setup.exe** becomes unavailable while it is running, the operating system sometimes displays a prompt to request that the end user insert the correct disk, and this may cause the installation to fail.) Therefore, to avoid this problem, the **Setup.exe** file is copied to a Temp folder, and the installation is relaunched from there. The original **Setup.exe** then terminates. However, when this happens, **DoInstall** (or **LaunchAppAndWait**, if it is called) behaves as if the installation has completed, and it does not wait.

Several workarounds for this issue exist. One method is to use the `/clone_wait` parameter when you are launching the child installation; as a result of this workaround, the launched installation keeps the original launched process running, and the parent installation then waits. Note, however, that this may cause problems if the original CD containing **Setup.exe** is not available throughout the entire installation. This includes multiple-CD installations, where the first CD is not available during some parts of the installation.

The only other way to avoid this problem is to add code that determines the ID of the child processes of the launched process and wait for the child process to complete.

Changes to Disk1 Files

InstallShield no longer places the following file on the Disk1 image of the built installation because it is no longer needed:

ISScript<version>.msi

In InstallShield 12 and later, the following file is placed on the Disk1 image of the built installation:

ISSetup.dll

The Disk1 file changes have not been known to cause any upgrade issues. These changes are reported for informational purposes.

Changes to the InstallScript Engine Files

With InstallShield 12 and later, InstallScript MSI installations no longer install InstallScript engine files to the following directory:

<COMMONFILES>\InstallShield\Professional\Runtime\<MajorVersion>\<MinorVersion>\Intel32

For InstallShield 11.5 and earlier InstallScript MSI projects, several files were stored in the **Binary** table:

- **Setup.inx**
- **Isconfig.ini**
- **Isrt.dll**
- **ISScriptBridge.dll**
- **_isresXXXX.dll** (where XXXX is the language—one .dll was included for each language included in the installation)
- **StringxXXXX.txt** (where XXX is the language—one .txt was included for each language in the installation)

For InstallShield 12 and later, all of these files (except **ISScriptBridge.dll**, which is no longer used) are stored inside the **ISSetup.dll** file, and that is the only file that is stored in the **Binary** table.

The InstallScript engine file changes have not been known to cause any upgrade issues. These changes are reported for informational purposes.

InstallScript-Related Custom Actions

Several built-in InstallScript-related custom actions were eliminated in InstallShield 12 for InstallScript MSI projects. If you upgrade an InstallScript MSI project from InstallShield 11.5 or earlier to InstallShield 2016, InstallShield removes these built-in custom actions from your project. Because these custom actions have been removed, some existing InstallScript event handler functions that were previously called by the custom actions are now called directly from InstallScript. For more information, see Creating and Scheduling InstallScript Custom Actions that Call InstallScript Event Handlers for InstallScript MSI Projects.

Changes for Deploying an InstallScript MSI Installation Without a Setup.exe File

To allow InstallScript MSI installations to be run without the **Setup.exe** file so that the packages can be deployed through technologies such as Active Directory, you may be able to use the instructions that are described in Knowledge Base article Q108166 - HOWTO: Deploying an MSI Wrapped with an InstallShield Script-Based Setup.exe. However, if you follow the procedure that is described in that article and users launch the .msi file directly, some of the InstallScript events (such as OnMoved) are never called. The reason that they are not called is that some of the InstallScript event handler functions that were previously called by built-in InstallScript-related custom actions are not called, since the InstallScript-related custom actions have been removed.

For more information, see Creating and Scheduling InstallScript Custom Actions that Call InstallScript Event Handlers for InstallScript MSI Projects.

Creating and Scheduling InstallScript Custom Actions that Call InstallScript Event Handlers for InstallScript MSI Projects

Several built-in InstallScript-related custom actions were eliminated in InstallShield 12 for InstallScript MSI projects. If you upgrade an InstallScript MSI project from InstallShield 11.5 or earlier to InstallShield 2016, InstallShield removes these built-in custom actions from your project. Because these custom actions have been removed, some existing InstallScript events that were previously called by the custom actions are now called directly from InstallScript. This article identifies the eliminated custom actions and explains how add custom actions that call these InstallScript events so that the scheduling is similar to the scheduling used in InstallShield 11.5 and earlier.

The eliminated custom actions also may affect your InstallScript MSI installation if you allow the installation to run without the **Setup.exe** file so that the packages can be deployed through technologies such as Active Directory. The procedure for implementing this is described in Knowledge Base article Q108166 - HOWTO: Deploying an MSI Wrapped with an InstallShield Script-Based Setup.exe. Note that if you follow the procedure that is described in that article and end users launch the .msi file directly, some of the InstallScript events are never called. The reason that they are not called is that some of the InstallScript event handler functions that were previously called by built-in InstallScript-related custom actions are not called, since the InstallScript-related custom actions have been removed. If you configure your installation according to the Q108166 article, you may also need to follow the instructions below to add custom actions that call the built-in InstallScript events.

The following built-in InstallScript-related custom actions were eliminated in InstallShield 12. They are removed from InstallScript MSI projects when they are upgraded from InstallShield 11.5 or earlier to InstallShield 2016:

- ISCleanupSuccess
- ISCleanUpSuspend
- ISCleanUpUserTerminate
- ISCleanupFatalExit
- ISMsiServerStartup
- ISRebootPatchHandler
- ISRollbackCleanup
- ISStartup
- OnCheckSilentInstall (For details about changes for this custom action, see OnCheckSilentInstall.)
- OnFeaturesInstalled
- OnFeaturesInstalling
- OnInstallFilesActionAfter
- OnInstallFilesActionBefore
- OnMoved
- OnMoving

Because these custom actions have been removed, some existing InstallScript event handlers that were previously called by the custom actions are now called directly from InstallScript. This includes the following InstallScript event handlers that are called immediately before file transfer:

- Feature functions for Installing and Uninstalling

- OnGeneratedMSIScript
- OnGeneratingMSIScript
- OnInstallFilesActionBefore
- OnMoving

It also includes the following InstallScript event handlers that are called immediately after file transfer:

- Feature functions for Installed and Uninstalled
- OnMoved
- OnInstallFilesActionAfter

These changes were made to make global variables and global pointers available for these InstallScript event handler functions. However, because previously the events were called from custom actions, the sequence of these events in relation to other custom actions that are called directly by Windows Installer has changed. Therefore, you may need to adjust your InstallScript code appropriately to account for these changes.

Except in the case of the feature event handlers, it is also possible to add custom actions that call these event handler functions so that the scheduling is similar to the scheduling used in InstallShield 11.5 and earlier. However, global variables and global pointers are not maintained between calls, as discussed in the Global Variables, Global Pointers, and SUPPORTDIR section.

Calling OnFeatureInstalling, OnFeatureUninstalling, OnFeatureInstalled, or OnFeatureUninstalled from a custom action during file transfer does not have any effect on the installation. This is because these functions are called from InstallScript just before file transfer, and they do not have any effect when they are called more than once.



Task ***To manually schedule an InstallScript custom action that calls a predefined InstallScript event handler function:***

1. Open the upgraded project in InstallShield 2016.
2. Make the appropriate changes to your InstallScript file:
   a. In the View List under **Behavior and Logic**, click **InstallScript**.
   b. Find the event handler function in your script. You can quickly find a function by clicking the function name in the center pane of the view.
   c. Rename the function to an alternate name to avoid conflicts with existing function prototypes automatically included in ifx.h. For example:
      - MyOnGeneratingMSIScript
      - MyOnMoving
      - MyOnMoved
   d. Update the existing function to take a single HWND parameter. For example:

      ```
      function MyOnGeneratingMSIScript(hMSI) begin end;
      ```

   e. Add appropriate prototypes for this new function:

      ```
      export prototype MyOnGeneratingMSIScript(HWND);
      ```

3. Add an InstallScript custom action that calls your renamed InstallScript event handler function:

   a. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.

   b. In the center pane, right-click the **Custom Actions** explorer and then click **New InstallScript**. InstallShield adds an InstallScript custom action.

   c. Type a name for the custom action; use the same name that you used to rename the InstallScript function. For example:

      - MyOnGeneratingMSIScript
      - MyOnMoving
      - MyOnMoved

   d. Select the new InstallScript custom action that you created.

   e. Set the **Function Name** setting to the name of the InstallScript function.

   f. For the **In-Script Execution** setting, specify the value that is indicated in the table below.

4. Schedule the InstallScript custom action for the appropriate part of the installation:

   a. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.

   b. Right-click the action or dialog that you want your InstallScript custom action to follow and click **Insert**. The **Insert Action** dialog box opens, providing a list of all the actions and dialogs that are currently associated with your project.

   c. Select the InstallScript custom action that you created. If appropriate, enter a condition in the **Condition** box for launching the InstallScript event.

   d. Click **OK**.

To match the previous functionality as closely as possible, set the in-script execution in step 3f and schedule the InstallScript custom actions in step 4b as follows:

Table 2-11 • Scheduling the Custom Actions

Custom Action	In-Script Execution	Location in the Sequence
OnGeneratingMSIScript	Immediate (default)	In the Installation Execute sequence between the RemoveExistingProducts and InstallInitialize actions
OnMoving	Deferred in system context	In the Installation Execute sequence between the InstallInitialize and AllocateRegistrySpace actions
OnFeaturesInstalling, which calls all the feature Installing and Uninstalling events that are defined. Note: This will not have any effect, as explained above.	Deferred in system context	In the Installation Execute sequence between the InstallInitialize and AllocateRegistrySpace actions (after OnMoving)

Table 2-11 • Scheduling the Custom Actions (cont.)

Custom Action	In-Script Execution	Location in the Sequence
OnInstallFilesActionBefore	Deferred in system context	In the Installation Execute sequence between the MoveFiles and InstallFiles actions
OnFeaturesInstalled, which calls all the feature Installed and Uninstalled events that are defined. Note: This will not have any effect, as explained above.	Deferred in system context	In the Installation Execute sequence between the ScheduleReboot and InstallFinalize actions (before OnMoved)
OnMoved	Deferred in system context	In the Installation Execute sequence between the ScheduleReboot and InstallFinalize actions (before OnGeneratedMSIScript)
OnInstallFilesActionAfter	Deferred in system context	In the Installation Execute sequence between the InstallFiles and PatchFiles actions
OnGeneratedMSIScript	Immediate (default)	In the Installation Execute sequence between the ScheduleReboot and InstallFinalize actions (after OnMoved)

Note the following:

- Global variables and pointers are no longer maintained between individual InstallScript custom action calls. In addition, each InstallScript custom action initializes and uses its own individual **SUPPORTDIR**. Therefore, you cannot share information across individual calls using files in **SUPPORTDIR**. For more information, see Global Variables, Global Pointers, and SUPPORTDIR.
- Most Windows Installer properties are not available to deferred, commit, and rollback InstallScript custom actions. For more information, see Windows Installer Properties and Deferred, Commit, and Rollback InstallScript Custom Actions.

OnCheckSilentInstall

In InstallShield 11.5 and earlier, the OnCheckSilentInstall custom action automatically called the OnMsiSilentInstall InstallScript event if the InstallScript MSI installation was being installed silently without using **Setup.exe** (for example, if the following command-line was used: `Msi.exe /i<Package> /qn`). In InstallShield 12 and later, this event

is not called in InstallScript MSI installations. However, you can add a custom action that calls the OnMsiSilentInstall event handler function. To do so, use the aforementioned instructions, noting the following scheduling requirements in steps 3f and 4b:

Table 2-12 • Scheduling the OnCheckSilentInstall Custom Action

Custom Action	In-Script Execution	Location in the Sequence
OnCheckSilentInstall	Immediate (default)	As the first action in the Installation Execute sequence

Also, to require that the InstallScript event run only in the expected scenario, add the following code to the event:

```
cchValueBuf = MAX_PATH;
// Check whether Setup.exe is being used, if so just return.
MsiGetProperty(nHandle, "ISSETUPDRIVEN", szValueBuf, cchValueBuf);
if(StrLengthChars(szValueBuf)) then
   return ISERR_SUCCESS;
endif;
```

Upgrading InstallShield 11.5 or Earlier InstallScript Projects

The following sections explain details about upgrading InstallScript projects that were created in InstallShield 11.5 or earlier to InstallShield 2016.

Including InstallScript Objects in InstallScript Projects

InstallScript installations created in InstallShield 2016 cannot consume objects that were created with InstallShield 11.5 or earlier.

If you upgrade an InstallScript project to InstallShield 2016, all references to any InstallShield objects are updated to point to the InstallShield 2016 versions of the objects. If the new version object is not present on the system, an error indicating that the object could not be found is displayed at build time.

For instructions on acquiring the InstallShield 2016 objects, see Obtaining Updates for InstallShield.

Changes to the DoInstall Function

The **DoInstall** function now launches the setup executable file of the child installation instead of using the already-running installation engine to run the child installation. This change should be automatic; that is, it should not require any changes to existing InstallScript code. The child installation may take slightly longer to initialize than in previous versions because the installation executable file needs to initialize. For more information, see DoInstall.

Note that now calling **DoInstall** is similar to calling **LaunchAppAndWait**. When the installation is run from any removable media, such as a CD-ROM or a DVD, the **Setup.exe** file on Disk1 may not be available during the entire installation. (If **Setup.exe** becomes unavailable while it is running, the operating system sometimes displays a prompt to request that the end user insert the correct disk, and this may cause the installation to fail.) Therefore, to avoid this problem, the **Setup.exe** file is copied to a Temp folder, and the installation is relaunched from there. The original **Setup.exe** then terminates. However, when this happens, **DoInstall** (or **LaunchAppAndWait**, if it is called) behaves as if the installation has completed, and it does not wait.

Several workarounds for this issue exist. One method is to use the `/clone_wait` parameter when you are launching the child installation; as a result of this workaround, the launched installation keeps the original launched process running, and the parent installation then waits. Note, however, that this may cause problems if the original CD containing **Setup.exe** is not available throughout the entire installation. This includes multiple-CD installations, where the first CD is not available during some parts of the installation.

The only other way to avoid this problem is to add code that determines the ID of the child processes of the launched process and wait for the child process to complete.

Setup.exe Changes

The `/deleter` command-line parameter for **Setup.exe** is no longer needed. If you specify this parameter, the installation will not run. Note that InstallScript installations no longer clone the installation immediately when they are launched (unless, for example, the installation is running from the temp folder because the `/runfromtemp` parameter is specified), so `/deleter` is no longer needed.

The `/clone_nowait` command-line parameter for **Setup.exe** is no longer needed. If you specify this parameter, **Setup.exe** ignores it. Note that InstallScript installations no longer wait for the cloned installation to complete by default unless the `/clone_wait` parameter is specified. For more information on `/clone_wait`, see Setup.exe and Update.exe Command-Line Parameters.

Like InstallScript MSI and Basic MSI installations, InstallScript's **Setup.exe** file returns meaningful return codes. Therefore, if you are checking the return value of **Setup.exe**, in InstallShield 11.5 and earlier, it would always be 0; in InstallShield 12 and later, **Setup.exe** returns an appropriate return value. For details on the possible return codes, see Setup.exe Return Values and Run-Time Errors (InstallScript Projects).

Changes to Disk1 Files

InstallShield no longer places the following files on the Disk1 image of the built installation because they are no longer needed:

- **Engine32.cab**
- **Setup.ibt**

In InstallShield 12 and later, the following files are placed on the Disk1 image of the built installation:

- **ISSetup.dll**
- **_Setup.dll**

The Disk1 file changes have not been known to cause any upgrade issues. These changes are reported for informational purposes.

Changes to the InstallScript Engine Files

With InstallShield 12 and later, InstallScript installations no longer install InstallScript engine files to the following directory:

<COMMONFILES>\InstallShield\Professional\Runtime\<MajorVersion>\<MinorVersion>\Intel32

This change has not been known to cause any upgrade issues. This change is reported for informational purposes.

Upgrading InstallShield 11.5 or Earlier QuickPatch Projects that Have InstallScript Custom Actions

QuickPatch projects that are created in InstallShield 12 and later for InstallScript MSI projects can only patch media that were also created with InstallShield 12 and later. If you want to use InstallShield 2016 to create an update for an application whose previous InstallScript MSI installation was created with InstallShield 11.5 or earlier, you should create a create a minor-upgrade package instead of a QuickPatch, and use the Patch Design view to create a standard patch. Subsequent patches from this version can be created as QuickPatch projects.

Creating a QuickPatch for a Basic MSI project (with or without InstallScript custom actions) has not been known to cause any issues.

Creating Standard Patches for InstallShield 11.5 and Earlier InstallScript MSI Projects

InstallShield 12 and later does not support the creation of InstallScript MSI patches (using the Patch Design view) where both the latest and previous setups were created with InstallShield 11.5 or earlier. Creating a patch where the latest setup is created with InstallShield 12 or later and the previous setup was created with InstallShield 11.5 or earlier has not been known to cause any issues.

Upgrading InstallShield 11.5 or Earlier InstallScript MSI Object Projects or Projects that Contain This Type of Object

InstallShield 11.5 and earlier included the InstallScript MSI Object type of project. This project type is no longer available in InstallShield.

Upgrading InstallScript MSI Object Projects

If you open an InstallScript MSI Object project in InstallShield 2016, InstallShield will convert it to a merge module project.

Predefined InstallScript event handlers are not available in merge module projects that have InstallScript custom actions. Therefore, if you have an InstallShield 11.5 or earlier InstallScript MSI object project that uses InstallScript event handler functions, you must manually schedule custom actions that call these functions when the project is converted to a merge module project in InstallShield 2016. To learn how, see Creating and Scheduling InstallScript Custom Actions that Call InstallScript Event Handlers for Basic MSI Projects.

Note that InstallShield 2016, as well as earlier and later versions of InstallShield, can consume InstallShield 2016 merge modules that have InstallScript custom actions.

Upgrading Projects that Contain InstallScript MSI Objects

Building upgraded projects that contain already-built InstallScript MSI objects (.imm files) will fail; these objects cannot be consumed in InstallShield 2016. If you try to do so, a build error such as the following one occurs:

```
ISDEV : error -4075: File not found. An error occurred merging Module
    'InstallShieldMSIObjectName.4F635B62_07BF_4779_B74E_D80C29D508E3:0' for Feature 'NewFeature1'.
```

where "InstallShieldMSIObjectName.4F635B62_07BF_4779_B74E_D80C29D508E3:0" is information that is specific to the missing InstallScript MSI object. To resolve this build error, remove this InstallScript MSI object from your project; you can do so by clearing its check box in the Redistributables view.

Upgrading Projects Created with InstallShield Express

If you have an InstallShield project that was created using versions 2.1 through 5.x of InstallShield Express, that project is automatically upgraded to a Basic MSI project when you open it in InstallShield 2016. InstallShield 2016 converts InstallShield Express projects to a Basic MSI project because it is the comparable project type.

To learn how to convert your upgraded project to an InstallScript MSI project, see Converting a Basic MSI Project to an InstallScript MSI Project.

Task ***To upgrade an Express project:***

Open the project in InstallShield. A dialog box opens to ask if you would like to upgrade your project.

- Click **Yes** to upgrade your project. Your project opens in InstallShield.
- Click **No** to leave the project unopened.

A backup copy of your original project is created and stored in the location specified in the dialog box.

InstallShield 2016 preserves the logic and dialog of the Setup Types view when it migrates Express 2.x projects (.iwz) to a Basic MSI project in InstallShield 2016. In earlier versions of Express, setup types were based on components. In Express 2.x projects (.iwz), components were replaced by features, and file groups are now mapped to destination folders instead of components.

When you upgrade an Express 2.x project (.iwz) to a Basic MSI project in InstallShield 2016, you can edit setup types by changing a feature's Install Level property and dialog control events.

Note • *Billboard properties from Express projects will not be migrated since they are not supported in Basic MSI projects. You will receive warning 701 when the billboards property is specified in earlier Express projects.*

If you migrate an Express project with a language that is not installed with your current version of InstallShield, you will lose support for that language. Language support is available in the Premier edition of InstallShield.

Upgrading Projects Created with InstallShield Professional

You can upgrade projects that were created using InstallShield Professional. When you are using InstallShield 2016 and you open an installation project that was created with InstallShield Professional, your project is automatically upgraded. This enables you to immediately work with your project in InstallShield 2016.

Projects that were created using InstallShield Professional were composed of several .ini files referenced by an .ipr file. In InstallShield 2016, your installation project is one file—an .ism file.

First Step: Opening Your Installation Project

Task ***To open an installation project created in InstallShield Professional:***

1. Open InstallShield.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.

3. Select the InstallShield Professional project file (.ipr) that you would like to open.
4. Click **Open**.

Your project opens in InstallShield, and the extension changes from .ipr to .ism. A backup of your original .ipr project is saved as *ProjectName*.ipr.bak in the same folder.

Next Steps: Modifying Your Installation Project

After you upgrade your installation project, you can use the views, menus, wizards, and other parts of the InstallShield user interface to modify your project settings.

You may also need to make changes to the upgraded project; see Migrating from InstallShield Professional 6.x and Migrating from InstallShield Professional 5.x.



Note • *In InstallShield 2016, support files are linked to a subfolder of the project folder; in InstallShield Professional, support files were physically copied to a subfolder of the project folder.*

Migrating from InstallShield Professional 6.x

When you upgrade an InstallShield Professional 6.x installation to InstallShield 2016, note the following:

- You can quickly convert a script that uses a program...endprogram block to an event-based script that calls all appropriate event handler functions except the user interface and file transfer functions. For details, see OnShowUI.
- The new simplified model for Internet installations eliminates many of the Ether object's methods and all of its properties and subobjects (while providing InstallScript enhancements that let you handle any Internet-specific requirements). If your Internet installation used any of these properties and methods, see Replacing Obsolete Properties and Methods.
- Installations created in version 6.x contained a number of automatically created string entries; the installation used these string entries to get information. These string entries were redundant—they contained the same information contained in the Project Settings property sheet. InstallShield 2016 simplifies matters by allowing the information specified in the project settings to be used at runtime.

  Note that using this information is not required; you can continue using your existing string entries as before by simply leaving them intact in your project.

  If you want your migrated installation to automatically use the information from your project's Project Settings property sheet, delete the following string entries from your project:

  COMPANY_NAME

  PRODUCT_NAME

  PRODUCT_KEY

  PRODUCT_VERSION

  TITLE_CAPTIONBAR

  FOLDER_NAME

  TITLE_MAIN

Also, update any script code that may be using one of these string entries to use the new corresponding system variable instead, as shown in the following table:

Table 2-13 • String Entries and Corresponding New System Variables

String Entry	System Variable
COMPANY_NAME	IFX_COMPANY _NAME
PRODUCT_NAME	IFX_PRODUCT_NAME
PRODUCT_KEY	IFX_PRODUCT_KEY
PRODUCT_VERSION	IFX_PRODUCT_VERSION
TITLE_CAPTIONBAR	IFX_SETUP_TITLE
FOLDER_NAME	IFX_PRODUCT_NAME
TITLE_MAIN	IFX_SETUP_TITLE

Note that you can remove only a subset of these string values if you want some values to be read from the string entries and some to be read from the project settings. If you retain any of these string values, be sure to modify them as required, rather than changing the unused project settings; in particular, when creating an update installation, be sure to update the value of the PRODUCT_VERSION string entry if it exists.

- If your installation uses an event-based script and you have overridden the default OnSetUpdateMode event handler code, a message such as the following may be displayed at run time: "The installed version of the application could not be determined. The setup will now terminate." If this error occurs, add code that manually sets IFX_INSTALLED_VERSION during the OnSetUpdateMode event. For sample code, see the default code that is included in the OnSetUpdateMode event of a new InstallShield 2016 InstallScript project.
- If your installation uses an event-based script and you have overridden the default OnFirstUIBefore event handler code, you must replace the following code (from the 6.x version of OnFirstUIBefore) in order to install files to the appropriate default location:

```
TARGETDIR = PROGRAMFILES ^ @COMPANY_NAME ^ @PRODUCT_NAME;
```

with the following code (from the new default OnFirstUIBefore code) to set TARGETDIR:

```
/* Handles both end users with administrative or power user privileges and
end users without such privileges. */
if ( ALLUSERS ) then
    TARGETDIR = PROGRAMFILES ^ IFX_COMPANY_NAME ^ IFX_PRODUCT_NAME;
else
    TARGETDIR = FOLDER_APPDATA ^ IFX_COMPANY_NAME ^ IFX_PRODUCT_NAME;
endif;

/* Handles both standard and multi-instance installations. */
if( MAINT_OPTION = MAINT_OPTION_MULTI_INSTANCE && MULTI_INSTANCE_COUNT > 0) then
    nLoop = 1;
    svDir = TARGETDIR;
    while(ExistsDir(TARGETDIR) = EXISTS)
        NumToStr(szTargetDirAppendix,nLoop);
        TARGETDIR = svDir + szTargetDirAppendix;
```

```
        nLoop = nLoop + 1;
    endwhile;
endif;
```

If you are using a procedural script (one with a program...endprogram block), you need to update the code in which you set TARGETDIR in order to set the default target directory appropriately for users without administrator privileges.

- If your installation uses an event-based script and you have overridden the default OnMaintUIBefore event handler code, and during uninstallation you want to remove all features including those that are not listed in the media but only in the log file (see FeatureRemoveAllInMediaAndLog for more information), then you must replace the following code (from the 6.x version of OnMaintUIBefore):

  ```
  case REMOVEALL: ComponentRemoveAll();
  ```

  with the following code (from the new default OnMaintUIBefore code):

  ```
  MediaGetData( MEDIA, MEDIA_FIELD_MEDIA_FLAGS, nMediaFlags, szIgnore );

  case REMOVEALL:
      /* Properly handles updating. */
      if( nMediaFlags & MEDIA_FLAG_UPDATEMODE_SUPPORTED ) then
          FeatureRemoveAllInMediaAndLog();
      else
          FeatureRemoveAllInMedia();
      endif;
  ```

  If you are using a procedural script that supports script-based uninstallation you need to update your call to ComponentRemoveAll appropriately.

- If your script calls RegDBSetItem to alter the registry entries that are created by MaintenanceStart (or DeinstallStart), and it makes those calls to RegDBSetItem in an event handler that is called before the OnMoved handler (for example, OnMoving or <feature name>_OnInstalled), you must move those calls to RegDBSetItem. MaintenanceStart is now called in the OnMoveData event handler's default code, whereas previously MaintenanceStart was called automatically immediately after the maintenance/uninstallation feature was installed and before any other features were installed.

- The InstallShield Professional 6.x default event handler code included the following lines in both OnFirstUIBefore and OnMaintUIBefore:

  ```
  SetStatusWindow( 0, "" );
  Enable( STATUSEX );
  StatusUpdate( ON, 100 );
  ```

  In InstallShield 2016, this code has been relocated to the default OnMoveData event handler code, so you have the following options:

  - If you have not customized this code, you do not need to do anything because the redundant calls will not cause a problem. You can safely remove this code from OnFirstUIBefore and OnMaintUIBefore if you want to avoid code duplication.
  - If you have customized this code and you want these customizations to apply regardless of what user interface (UI) event is called, you should remove the code from OnFirstUIBefore and OnMaintUIBefore, then override the OnMoveData event and place the customized code in place of the default code.
  - If you have customized the code and you want specific code depending on what UI event is called, you should override OnMoveData, comment out the default code, and continue to use your existing code. Note that in this case, if your installation supports updating you may also want to customize OnUpdateUIBefore.

- The 6.x default event handler code included the following commented-out code in the OnFirstUIBefore and OnMaintUIBefore events:

  ```
  // TO DO: if you want to enable background, window title, and caption bar title
  // SetTitle( @TITLE_MAIN, 24, WHITE );
  // SetTitle( @TITLE_CAPTIONBAR, 0, BACKGROUNDCAPTION );
  // Enable( FULLWINDOWMODE );
  // Enable( BACKGROUND );
  // SetColor( BACKGROUND, RGB( 0, 128, 128 ) );
  ```

  If you activated this code and would like to provide a consistent UI experience regardless of what UI event is called, you can remove this code from OnFirstUIBefore and OnMaintUIBefore and then override OnShowUI and customize the code appropriately.

- Many Windows API functions are declared in included header files (.h files) in InstallShield 2016. If any of these functions are also explicitly declared in your script code, do one of the following:

  - Remove your function declaration and use the declaration provided by InstallShield Professional.
  - If you are creating a script that needs to be compilable in both InstallShield 2016 and earlier versions, surround your declaration with code like the following:

    ```
    #if _ISCRIPT_VER < 0x700
    prototype ...
    #endif
    ```

  You may also need to update code that calls a Windows API function, if the declaration defined by InstallShield Professional is different than your declaration.

- In order that the end user interface flows smoothly, by default the installation initialization dialog box remains displayed until the script displays a dialog box. To close the installation initialization dialog box, call Disable(DIALOGCACHE) or any dialog box function.

Migrating from InstallShield Professional 5.x



Project • *This information applies to InstallScript Object projects.*

Script Changes

The basic structure of an InstallShield Professional 5.x script is supported in the InstallScript project type in InstallShield 2016. To compile a Professional 5.x script in InstallShield, you must first make the following changes:

Preprocessor Directives

- Include the following statement at the beginning of your script:

  ```
  #include "ifx.h"
  ```

- Remove any #define statements for SD_SINGLE_DIALOGS and its associated SD_<dialog name> constants.
- You may need to remove some #define statements that define Windows constants if those statements result in compiler errors. Many Windows constants are defined in **Ifx.h** (which is in the *`InstallShield Program Files Folder`***\Script\Isrt\Include** subfolder) or its included files.

Built-In Functions

- Remove the following functions, which were supported in Professional 5.x but are not supported in InstallShield 2016. Some of these functions will compile, but they will not give the expected results. Where possible, supported alternatives are suggested.

Table 2-14 • Obsolete Professional 5.x Functions

Professional 5.x Function	Suggested Alternative
AppCommand	None. (Appropriate only for platforms using the Program Manager shell, which are no longer supported.)
AddProgItemEx	AddFolderIcon
CmdGetMsg	Windows API function GetMessage (call CmdGetHwndDlg to get the dialog handle)
CmdGetParam1	Windows API function GetMessage (call CmdGetHwndDlg to get the dialog handle)
CmdGetParam2	Windows API function GetMessage (call CmdGetHwndDlg to get the dialog handle)
CommitSharedFiles	The actions performed by this function in Professional 5.x are done automatically in InstallShield 2016.
CreateProgGroupEx	AddFolderIcon
DeleteGroup	DeleteProgramFolder
DeleteProgItem	DeleteFolderIcon
ExitProgMan	None. (Appropriate only for platforms using the Program Manager shell, which are no longer supported.)
GetGroupNameList	GetFolderNameList
GetItemNameList	None. (Appropriate only for platforms using the Program Manager shell, which are no longer supported.)
QueryProgGroup	None. (Appropriate only for platforms using the Program Manager shell, which are no longer supported.)
RegDBCreateKey	RegDBCreateKeyEx
RegDBCreateKeyValue	RegDBCreateKeyValueEx
RegDBGetKeyValue	RegDBGetKeyValueEx
RegDBSetKeyValue	RegDBSetKeyValueEx

Table 2-14 • Obsolete Professional 5.x Functions (cont.)

Professional 5.x Function	Suggested Alternative
ReloadProgGroup	None. (Appropriate only for platforms using the Program Manager shell, which are no longer supported.)
ReplaceProgItem	None. (Appropriate only for platforms using the Program Manager shell, which are no longer supported.)
ShowGroup	None. (Appropriate only for platforms using the Program Manager shell, which are no longer supported.)

- Call Disable(LOGGING) before any code that makes changes to the target system that you do not want logged for uninstallation, and call Enable(LOGGING) after that code. This is necessary because logging of changes for uninstallation is automatically enabled before any script code is executed.

  Note that this is different from Professional 5.x installations in which logging did not begin until DeInstallStart was called. If you have any installation actions that previously were not logged because they occurred before DeInstallStart, you will need to add an additional Disable(LOGGING) call before these installation actions to disable logging manually.

  If you want your installation to display a background, you must call Enable(BACKGROUND). In InstallShield 2016, the background is disabled by default, whereas in Professional 5.x it was enabled by default. (In event-based scripts, code for customizing and displaying a background exists in comment lines at the beginning of the default code for OnFirstUIBefore. To activate this code, remove the slash characters at the beginnings of the desired lines.)

  Note that some user-interface functions, such as PlaceBitmap, will not work properly (nor return a negative value to let you know they failed) if the installation background is not enabled.

- It is no longer necessary to call RegDBSetItem with the argument REGDB_UNINSTALL_NAME to set the value of [DisplayName] under the uninstallation key. This is now done when MaintenanceStart or DeinstallStart is called.

- Calling SetColor with STATUSBAR as its argument will not have any effect in InstallShield 2016; status bars are drawn using system colors.

- When you call StrGetTokens, if the first (or last) character of szString matches a character in szDelimiterSet, a null string ("") is not inserted in the list as the first (or last) element. Instead, the characters between the first and second (or last and next to last) delimiters are inserted in the list as the first (or last) element.

- When you call StrGetTokens, if a wildcard character is included in the filename specified by szSrcFile, the filename part of szTargetFile is not ignored as in previous versions of InstallShield Professional; instead, all of szTargetFile is treated as the target path to which each source file is copied to with its existing name. For example, CopyFile("C:*.*", "D:\\File.txt") copies files to a folder named File.txt on the D drive.

- WaitOnDialog now returns IDCANCEL when the end user selects the system menu's Close command or clicks the title bar's Close button from a custom dialog. In previous versions, DLG_CLOSE was returned in these cases.

- If you pass GetSystemInfo a first argument of DRIVE and a third argument that is a UNC path, the function correctly returns a negative value. Previously it returned zero, falsely indicating success, although due to operating system limitations UNC paths are not supported in that case.

- When you call ParsePath with its third argument set to PATH and its second argument set to a UNC path that includes only a server name without a trailing backslash (for example, "\\\\TheServer"), the function correctly

sets the first argument equal to a double backslash ("\\\\"). In Professional 5.x, the function would set the first argument equal to a single backslash ("\\").

- The Professional 5.x Help Library listed specific numeric return values, other than 0 (zero) for success, for some functions. Some of these numeric values have changed for InstallShield 2016; a script used in InstallShield 2016 should compare a function's return value only to the constants that are currently listed in that function's help.
- In order for the end-user interface to flow smoothly, by default the installation initialization dialog remains displayed until the script displays a dialog. To close the installation initialization dialog, call Disable(DIALOGCACHE) or any dialog function.
- ComponentMoveData's second argument no longer returns useful data.
- Passing a pointer to a string as the fourth argument of SendMessage no longer gives the expected result. To pass string data to SendMessage, see the Additional Information section of the SendMessage topic.

Predefined Constants

- Remove the following constants, which were supported in Professional 5.x but are not supported in InstallShield 2016:

  COMPONENT_VALUE_ALWAYSOVERWRITE

  COMPONENT_VALUE_NEVEROVERWRITE

  COMPONENT_VALUE_NEWERDATE

  COMPONENT_VALUE_NEWERVERSION

  COMPONENT_VALUE_OLDERDATE

  COMPONENT_VALUE_OLDERVERSION

  COMPONENT_VALUE_SAMEORNEWDATE

  COMPONENT_VALUE_SAMEORNEWERVERSION

  COMPONENT_FIELD_DESTINATION

  COMPONENT_FIELD_OVERWRITE

- Remove statements that assign values to the following predefined variables (for example, TARGETDISK="C:\\";). Such statements will not compile in InstallShield 2016; these constants are now read-only.

  CMDLINE

  COMMONFILES

  ERRORFILENAME

  FOLDER_DESKTOP

  FOLDER_PROGRAMS

  FOLDER_STARTMENU

  FOLDER_STARTUP

  ISRES

  ISUSER

ISVERSION

MODE

PROGRAMFILES

SUPPORTDIR

TARGETDISK (The value of this variable is automatically updated when TARGETDIR is changed.)

UNINST

WINDIR

WINDISK

WINSYSDIR

WINSYSDISK

DLLs

- When you called a DLL function in Professional 5.x, all string arguments were passed by reference; in InstallShield 2016, you can pass a string argument by value. To specify the method for passing an argument, in the function prototype, precede the argument's data type with the BYREF or BYVAL keyword; for example, MyDLL.MyFunc(BYREF NUMBER, BYVAL STRING). The InstallShield 2016 compiler includes the following compiler warnings to encourage the use of the appropriate keyword.
  - If your script includes DLL function calls where neither BYREF nor BYVAL has been specified, you receive compiler warning W7507. To eliminate this warning, add the appropriate keyword to the function prototype.
  - If you specify a literal string for a string argument that is passed by reference, you receive compiler warning W7511. To eliminate this warning, add the BYVAL keyword to the function prototype.

  When you call a function in an external DLL, make sure the same calling convention is used in the script and the DLL. In Professional 5.x, the installation engine always used the stdcall convention but would sometimes overlook an inconsistent DLL convention. The InstallShield 2016 engine does not; it generates an error (exception 0x80040704) when the wrong calling convention is used.

  In your script, you can declare a calling convention, either cdecl or stdcall, when declaring a DLL function. For example:

  prototype cdecl MyDLL.MyFunction (INT, INT);

  If you do not explicitly declare a calling convention, InstallShield uses stdcall.
- In InstallShield 2016, InstallScript arrays are internally formatted as OLE Automation safe arrays, not native C or C++ arrays. To pass an InstallScript array to a DLL function that expects a C or C++ array, you must place the array data in the expected format by calling GetCArrayFromISArray.
- In InstallShield 2016, a string variable whose address is stored in a pointer variable cannot be changed by passing the pointer to a DLL function. To allow a DLL function to change the value of a string variable, pass the variable itself as an argument to the function, after declaring the data type of that argument specifying the BYREF operator.
- You no longer need to call the UseDLL function before calling a DLL that is located on Windows' DLL search path. The DLL search path is the following:

1. The folder from which the application loaded.
2. The current folder.
3. The 32-bit Windows system folder.
4. The 16-bit Windows system folder.
5. The Windows folder.
6. The folders that are listed in the PATH environment variable.

Use this feature with caution; the values of the current folder and PATH environment variable on an end user's machine are not predictable.

Miscellaneous

- The undocumented array syntax ArrayName[nArrayIndex] is no longer supported. In InstallShield 2016, the syntax for all arrays is ArrayName(nArrayIndex).
- If you explicitly declare a return type for a script-defined function, you must specify the same return type in the function definition, after the keyword "function", or the script will not compile. (If you do not explicitly declare a return type, the return type is assumed to be NUMBER.) In previous versions of InstallShield Professional, return types were assumed to be NUMBER regardless of explicit specifications of other return types, so this issue did not arise.
- To place uninstall information under HKEY_CURRENT_USER, set the value of the ALLUSERS system variable to FALSE.
- To pass a literal character as an argument to a built-in or user-defined function, you must pass its numeric ASCII value.
- When processing the system variable CMDLINE, be aware that the following command line switches are no longer used by Setup.exe and so are included in CMDLINE: -SMS, -z, -c, -e, -q, -t, and -x. The -SMS switch is obsolete because now Setup.exe always stays in memory until the installation is complete. The -z switch is obsolete because now Setup.exe initializes correctly even on systems with more than 256 megabytes of memory.
- Remove Professional 5.x's ODBC Template code, which will not compile in InstallShield 2016. This is because the following function call is nested within the code:

```
ComponentGetData ( szMedia, szCompName,
   COMPONENT_FIELD_DESTINATION, nvData, svDest );
```

In InstallShield 2016 this function call does not compile because the Destination property is a file group property, not a component property.

Other Changes

- Pressing F3 no longer works to exit an installation.
- Setup.exe no longer supports the -f switch for using a renamed Setup.inx.
- InstallShield 2016 installations do not support including special billboards for low-resolution systems.
- If neither a source file nor the already-existing target file has a version number, and the source file's file group's Overwrite property is "Newer Version", "Same or Newer Version", or "Older Version", then the file on the target system is not overwritten. In Professional 5.x installations, the target file would be overwritten.

- InstallShield 2016 does not support displaying Windows 95-style dialogs.
- The Lang key in a silent installation's response (.iss) file's [Application] section no longer has any effect.
- In Professional 5.x installations, any registry key logged by the installer would be removed during uninstallation in any case. In InstallShield 2016, keys created by RegDBCreateKeyEx and non-shared keys in registry sets are removed only if the installer created the key, that is, the key did not exist when the installation was run.

Script Changes: Lexicon Conversion

Terminology Change

In InstallShield Professional, the functional building blocks of your installation from the end user's perspective were called components. With InstallShield 2016, these building blocks are called features.

Because of this change in terminology, a number of InstallScript function names have also changed. When you upgrade a project that you created using InstallShield Professional, some items in your script might undergo a lexicon change or be aliased.

- Function Names
- ComponentFileInfo (FeatureFileInfo) Flags
- ComponentSetData (FeatureSetData) and ComponentGetData (FeatureGetData) Flags

Function Names

Any InstallScript function name that referred to component now refers to feature. For example, ComponentDialog is now FeatureDialog. The parameters for these functions have not changed. Click a feature function to view the corresponding help topic.

Table 2-15 • Function Name Changes

Component Function	Feature Function
ComponentAddItem	FeatureAddItem
ComponentCompareSizeRequired	FeatureCompareSizeRequired
ComponentDialog	FeatureDialog
ComponentError	FeatureError
ComponentErrorInfo	FeatureErrorInfo
ComponentFileEnum	FeatureFileEnum
ComponentFileInfo	FeatureFileInfo
ComponentFilterLanguage	FeatureFilterLanguage
ComponentFilterOS	FeatureFilterOS

Table 2-15 • Function Name Changes (cont.)

Component Function	Feature Function
ComponentGetData	FeatureGetData
ComponentGetItemSize	FeatureGetItemSize
ComponentGetTotalCost	FeatureGetTotalCost
ComponentInitialize	FeatureInitialize
ComponentIsItemSelected	FeatureIsItemSelected
ComponentListItems	FeatureListItems
ComponentLoadTarget	FeatureLoadTarget
ComponentMoveData	FeatureMoveData
ComponentPatch	FeaturePatch
ComponentReinstall	FeatureReinstall
ComponentRemoveAll	FeatureRemoveAll
ComponentRemoveAllInLogOnly	FeatureRemoveAllInLogOnly
ComponentRemoveAllInMedia	FeatureRemoveAllInMedia
ComponentRemoveAllInMediaAndLog	FeatureRemoveAllInMediaAndLog
ComponentSaveTarget	FeatureSaveTarget
ComponentSelectItem	FeatureSelectItem
ComponentSelectNew	FeatureSelectNew
ComponentSetData	FeatureSetData
ComponentSetTarget	FeatureSetTarget
ComponentSetupTypeEnum	FeatureSetupTypeEnum
ComponentSetupTypeGetData	FeatureSetupTypeGetData
ComponentSetupTypeSet	FeatureSetupTypeSet
ComponentTotalSize	FeatureTotalSize
ComponentTransferData	FeatureTransferData

Table 2-15 • Function Name Changes (cont.)

Component Function	Feature Function
ComponentUpdate	FeatureUpdate
ComponentValidate	FeatureValidate
SdComponentDialog	SdFeatureDialog
SdComponentDialogAdv	SdFeatureDialogAdv
SdComponentMult	SdFeatureMult
SdComponentTree	SdFeatureTree

ComponentFileInfo (FeatureFileInfo) Flags

Flags that you used with the ComponentFileInfo (now FeatureFileInfo) function in InstallShield Professional are converted to reflect their use with features in InstallShield 2016.

Table 2-16 • ComponentFileInfo Flags

Deprecated Flag	New Flag
COMPONENT_INFO_ATTRIBUTE	FEATURE_INFO_ATTRIBUTE
COMPONENT_INFO_LANGUAGE	FEATURE_INFO_LANGUAGE
COMPONENT_INFO_OS	FEATURE_INFO_OS
COMPONENT_INFO_ORIGSIZE	FEATURE_INFO_ORIGSIZE
COMPONENT_INFO_COMPSIZE **COMPONENT_INFO_DATE** **COMPONENT_INFO_DATE_EX** **COMPONENT_INFO_TIME**	***Note*** • *The flags that are not available return -137, which indicates that the option is not functional. You can use the FeatureError function to provide more information about the return value.*
COMPONENT_INFO_VERSIONLS	FEATURE_INFO_VERSIONLS
COMPONENT_INFO_VERSIONMS	FEATURE_INFO_VERSIONMS
COMPONENT_INFO_VERSIONSTR	FEATURE_INFO_VERSIONSTR

ComponentSetData (FeatureSetData) and ComponentGetData (FeatureGetData) Flags

Flags that you used with the ComponentSetData and ComponentGetData (now FeatureSetData and FeatureGetData) functions in InstallShield Professional are converted to reflect their use with features in InstallShield 2016.

Table 2-17 • Deprecated ComponentSetData and ComponentGetData Flags

Deprecated Flag	New Flag
COMPONENT_FIELD_CDROM_FOLDER	FEATURE_FIELD_CDROM_FOLDER
COMPONENT_FIELD_DESCRIPTION	FEATURE_FIELD_DESCRIPTION
COMPONENT_FIELD_DISPLAYNAME	FEATURE_FIELD_DISPLAYNAME
COMPONENT_FIELD_FILENEED	FEATURE__FIELD_FILENEED
COMPONENT_FIELD_FTPLOCATION	FEATURE_FIELD_FTPLOCATION
COMPONENT_FIELD_HTTPLOCATION	FEATURE_FIELD_HTTPLOCATION
COMPONENT_FIELD_IMAGE	FEATURE_FIELD_IMAGE
COMPONENT_FIELD_MISC	FEATURE_FIELD_MISC
COMPONENT_FIELD_PASSWORD	FEATURE_FIELD_PASSWORD
COMPONENT_FIELD_SELECTED	FEATURE_FIELD_SELECTED
COMPONENT_FIELD_SIZE	FEATURE_FIELD_SIZE
COMPONENT_FIELD_STATUS	FEATURE_FIELD_STATUS
COMPONENT_FIELD_VISIBLE	FEATURE_FIELD_VISIBLE
COMPONENT_VALUE_CRITICAL	FEATURE_VALUE__CRITICAL
COMPONENT_VALUE_HIGHLYRECOMMENDED	FEATURE_VALUE__HIGHLYRECOMMENDED
COMPONENT_VALUE__STANDARD	FEATURE_VALUE_STANDARD

Adding Project Assistant Dialog Support to Projects Upgraded from InstallShield Professional

When you upgrade an installation project from InstallShield Professional to InstallShield and the project contains a Setup.rul file that was created in InstallShield Professional, some questions may not be available on the Installation Interview page of the Project Assistant. This is because the Setup.rul file is missing the portions of the OnFirstUIBefore event code that InstallShield uses to key on script tags.

If you create a new InstallScript project and look at the OnFirstUIBefore event, you will see the following code that the Project Assistant requires in order to support dialog functions:

```
//{{IS_SCRIPT_TAG(FunctionName)
    InstallScript code...
//}}IS_SCRIPT_TAG(FunctionName)
```



Task ***To add Project Assistant dialog support to your upgraded project:***

1. In the **InstallScript** view, open your Setup.rul file.
2. Comment out the OnFirstUIBefore event code.
3. In the event category list at the top of the InstallScript pane, select the OnFirstUIBefore event. InstallShield re-adds this event to your script.
4. Copy the pertinent dialog code to the commented-out code for the OnFirstUIBefore event.
5. Delete the OnFirstUIBefore event code that you added in step 3.
6. Remove the comments from the remaining OnFirstUIBefore event code.

Upgrading Projects Created with InstallShield—Windows Installer Edition

If you have an InstallShield project (.ism file) that was created using InstallShield—Windows Installer Edition, that project is automatically upgraded to a Basic MSI project when you open it in InstallShield. See Converting a Basic MSI Project to an InstallScript MSI Project to learn how to convert your upgraded project to an InstallScript MSI project.



Task ***When you open the project, a dialog box appears to ask if you would like to upgrade your project.***

- *Click **Yes** to upgrade your project. Your project opens in the IDE.*
- *Click **No** to leave the project unopened.*

A backup copy of your original project is created and stored in the location specified in the dialog.

Upgrading a Setup from the Command Line

You can also upgrade your setup project when you build it from the command line. Use the -u parameter to upgrade your setup project from the command line without building it.

Troubleshooting Upgrade Errors

Any errors that result from the upgrade are displayed in the output panel at the bottom of the IDE. For information on these errors, see Upgrade Errors and Warnings (Upgrading from InstallShield—Windows Installer Edition).

Post-Upgrade Changes in the Property Manager

Properties in the Property Manager with an empty value (no value) are invalid for MSI. Because of this, ***DO_NOT_BUILD*** appears in place of the empty value for these properties. In this way, properties with empty values are preserved as placeholders after the upgrade, but are not built into the .msi package.

Upgrading from Other InstallShield Editions

InstallShield is available in three different editions: Premier, Professional, and Express.

Features that Are in Only the Premier Edition

Following is a list of some of the features that are available in the Premier edition but not the Professional, or Express editions:

- **Ability to create and build Suite/Advanced UI installations**—Create a bootstrap application with a contemporary, customizable user interface for multiple .msi packages, .msp packages, InstallScript packages, .exe packages, sideloading UWP app packages (.appx), and Windows Installer transactions, as well as multiple InstallShield prerequisites. A Suite/Advanced UI installation packages together multiple separate installations as a single installation while providing a unified user interface; it uses a setup launcher (**Setup.exe**) to conditionally launch packages on target systems as needed.
- **Support for InstallScript actions and other types of actions in Suite/Advanced UI installations**—Suite/Advanced UI installations have built-in support for launching different types of actions to perform various run-time tasks that are outside the scope of the packages that you are including in the installation. The actions can run executable files, call DLL functions, run PowerShell scripts, set a Suite/Advanced UI property, run InstallScript code, or call a public method in a managed assembly at run time.
- **Virtualization support**—The Microsoft App-V Assistant is included in the Premier Edition of InstallShield. Use this assistant to create customized virtual applications in the Microsoft App-V format. Virtualization enables you to isolate an application in its own environment so that it does not conflict with existing applications or modify the underlying operating system.
- **Application Virtualization Suitability Suites**—Several validation suites are available in InstallShield for helping you to determine how ready your products are for virtualization. The InstallShield virtualization internal consistency evaluators (ISVICEs) that are included in these suites let you check suitability for Microsoft App-V 4.x, Microsoft App-V 5.x, Microsoft Server App-V, VMware ThinApp, and Citrix XenApp. The validation suites can enable you to make more informed decisions about how you should build your product if you are considering offering your customers a virtualized version.
- **Support for DIM files**—The ability to create DIM projects is available in the Premier edition of InstallShield. This support is also available in the InstallShield Developer Installation Manifest Editor, a collaboration add-on. The ability to add DIM files to Basic MSI projects is available in the Premier edition of InstallShield.

  A DIM project is a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete portion of a product installation. Working with DIMs enables multiple team members to contribute to the development of the installation simultaneously. Each software developer or other team member can work on a separate DIM that the release engineer can reference in one or more Basic MSI projects.

- **Multilingual installations**—Create a single installation that displays end-user text in multiple languages and can handle conditional installation of language-specific files. Change dialogs and messages to any one of 34 additional languages using pre-translated strings.

  Project • *Note that support for two of those languages—Arabic (Saudi Arabia) and Hebrew—is available in only Basic MSI and Merge Module projects.*

- **Arabic (Saudi Arabia) and Hebrew language support**—InstallShield Premier Edition includes support for Arabic (Saudi Arabia) and Hebrew languages, which are written and read from right to left. All of the default end-user dialog strings are available in these languages.

  Since these languages are read from right to left, the Premier edition also includes support for mirroring Arabic and Hebrew dialogs; that is, InstallShield uses a right-to-left layout for Arabic and Hebrew dialogs. Thus, for example, buttons that are on the right side of dialogs in English and other left-to-right languages are moved to the left side of right-to-left-language dialogs.

- **Ability to specify commands that run before, during, and after builds**—InstallShield Premier Edition includes release settings that you can use to specify commands that you want to be run at various stages of the build process. You can schedule commands that run at the following build events: (a) before InstallShield starts building the release, (b) after InstallShield has built the .msi package and the .cab files (if your product's data files are to be stored in .cab files), but before the .msi package has been digitally signed and streamed into the **Setup.exe** file, and (c) after InstallShield has built and signed the release.

- **Ability to distribute installations to virtual machines that InstallShield provisions at build time or on demand**—You can configure your projects so that after each successful build of your installation, InstallShield automatically reverts a virtual machine (VM) to a designated snapshot, powers on the VM, and copies your installation to the VM to make it available for testing. You can also alternately perform these testing preparation steps on demand at any time. The testing preparation capability makes it possible to reduce testing time and eliminate manual steps. The VM can be on a Microsoft Hyper-V Server, a VMware ESX or ESXi Server, or a VMware Workstation.

- **Extra licenses for InstallShield MSI tools**—InstallShield includes several tools: InstallShield MSI Diff, InstallShield MSI Query, InstallShield MSI Sleuth, and InstallShield MSI Grep. You can use these tools to troubleshoot issues with Windows Installer packages. InstallShield Premier Edition includes a separate installation and extra licenses that let you install just the InstallShield MSI tools, without InstallShield, on separate machines. For specific terms, see the End-User License Agreement for the InstallShield MSI tools.

- **Ability to import IIS data from existing IIS Web sites into a project**—InstallShield includes an IIS scanner (**IISscan.exe**), a command-line tool for scanning an existing IIS Web site and recording IIS data about the Web site. The IIS scanner creates an XML file that contains all of the settings for the Web site, its virtual directories, its applications, and its application pools. You can use the XML file to import the IIS data into the Internet Information Services view in InstallShield Premier Edition. Once you have imported the IIS data into a project, you can use the Internet Information Services view to make changes to the IIS settings as needed.

- **Support for deploying Web Deploy packages**—Suite/Advanced UI installations, available in the Premier edition of InstallShield, have built-in support for deploying Web Deploy packages to IIS Web servers and the cloud.

- **InstallShield Collaboration**—InstallShield Premier Edition includes licenses for InstallShield Collaboration for Visual Studio.

- **Network repository**—A network repository is a collection of installation elements that multiple installation authors can access and reuse in their projects as needed. A network repository fosters collaboration among installation authors; it is stored on a network.
- **InstallShield Best Practice Suite**—InstallShield includes a set of validators called the InstallShield Best Practice Suite. The InstallShield Best Practice (ISBP) validators in this suite alert you if your installation violates best-practice guidelines.
- **Additional dialog themes**—Several dialog themes are available only in the Premier edition of InstallShield.

Features that Are in Only the Premier and Professional Editions

Following is a list of some of the features that are available in the Premier and Professional editions but not the Express edition:

- **Ability to create and build Advanced UI installations**—Create a bootstrap application with a contemporary, customizable user interface for a single .msi package, .msp package, or InstallScript package, as well as multiple InstallShield prerequisites. An Advanced UI installation uses a setup launcher (**Setup.exe**) to conditionally launch packages on target systems as needed.

  (Note that the Suite/Advanced UI functionality in the Premier edition of InstallShield includes extended support for this functionality: The Suite/Advanced UI functionality enables you to package multiple .msi packages, .msp packages, InstallScript packages, .exe packages, sideloading UWP app packages (.appx), and Windows Installer transactions, as well as multiple InstallShield prerequisites into a single installation with a contemporary, unified, customizable user interface.)
- **Standalone Build**—This tool, which is available with the Premier and Professional editions of InstallShield, enables you to install only the part of InstallShield that builds the installations, plus any redistributables that you want to include, on a build machine. Extra licenses of the Standalone Build are available for purchase.
- **Dialog Editor**—The Dialog Editor enables you to modify the layout of existing end-user dialogs or create new custom dialogs. Import and export dialogs to share them across projects. Construct different dialogs for each language supported in the project.
- **InstallShield MSI tools**—The Premier and Professional editions of InstallShield includes several tools: InstallShield MSI Diff, InstallShield MSI Query, InstallShield MSI Sleuth, and InstallShield MSI Grep. You can use these tools to troubleshoot issues with Windows Installer packages.
- **Automation interface**—Use script to add new files, add or delete features, change the product name and upgrade code, change release settings, change summary information stream items, change release flags, change any property, initiate the build process, and more.
- **Release customization**—Define which project segments to compress, which features to place on which disk, and which languages to include. Choose to filter application data based on language to support localization efforts.
- **Source code control integration**—Simplify the process of checking projects in and out of your source code control system and save space when differencing projects. The SCC integration in the Premier and Professional editions of InstallShield supports integration with various source code control systems.
- **Flexible localization support**—The Premier and Professional editions of InstallShield include a String Editor view, which gives you complete and centralized control over the localizable text strings that are displayed at run time during the installation process. You can use this view to edit the strings for everything from button text to feature descriptions. This view also includes support for exporting string entries (which you can have translated) and for importing translated string entries back into your project.

- **Project validation**—Use standard .cub files to validate installations and merge modules. Use the upgrade and patching validation to find out about potential upgrade problems and resolve them before you release your upgrades and patches.
- **Patch creation**—In addition to enabling you to create QuickPatch projects, the Premier and Professional editions let you create standard patches that contain updates to a previous version of your product.
- **Manage multiple product versions**—Build versions such as Evaluation, Debug, Standard, and Advanced—from a single project. Allow specific features, InstallShield prerequisites, and other elements to be chosen for inclusion in (or exclusion from) a release through user-defined flags.
- **Support for InstallScript**—The Premier and Professional editions of InstallShield include support for InstallScript, a simple but powerful programming language. You can add InstallScript custom actions to Windows Installer–based installations or create InstallScript projects, which use the InstallScript engine instead of the Windows Installer engine to control the entire installation.
- **Flexible custom action support**—The Premier and Professional editions of InstallShield include support for several custom action types that are not available in the Express edition. These extra custom action types enable you to do the following: set a property, set a directory, call a public method in a managed assembly, or display error message under certain conditions and abort the installation.
- **Flexible shortcut support**—The Premier and Professional editions of InstallShield include support for configuring various advanced settings for shortcuts that are not configurable in the Express edition. For example, the Premier and Professional editions let you prevent a shortcut in your project from being pinned to the taskbar or to the Start menu. They also let you prevent a shortcut on the Start menu from being highlighted as newly installed after end users install your product. These shortcut options are often used for tools or secondary products that are part of an installation.
- **Merge module authoring and editing**—Package pieces of a project for reuse across application installations. Reuse those you create or any of the ones included in the product. Edit and open modules for greater customization.
- **Project templates**—Create project templates that contain all of the default settings and design elements that you want to use as a starting point when you create an installation project or merge module project.
- **Multiple IIS Web sites**—The Express and Limited editions of InstallShield let you install only one Web site per installation. The Premier and Professional editions let you install more than one Web site per installation.
- **Support for IIS application pools and Web service extensions**—Install and manage IIS application pools and Web service extensions.
- **Advanced support for Windows services**—Although the Express edition of InstallShield includes some support for working with services, the Premier and Professional editions of InstallShield include additional flexibility for services. For example, the Premier and Professional editions enable you to start, stop, or delete a service during installation or uninstallation; the service can be part of your installation, or it can be already present on target systems. These editions also let you configure extended service customization options that were introduced in Windows Installer 5.
- **SQL support**—Connect to SQL servers, import database schema and data, associate SQL scripts with features, and more with SQL support.
- **Ability to modify text files or XML files**—Use the Text File Changes view or the XML File Changes view to configure files that you want to modify on the target system at run time.
- **Pure 64-bit support**—Windows Server Core supports disabling 32-bit Windows-on-Windows (WOW64) support. As this configuration becomes more popular, you may want to ensure that your 64-bit applications can

install without any reliance on 32-bit functionality. To make this possible, the Premier and Professional editions of InstallShield include support for building pure 64-bit .msi packages; these can be run on 64-bit Windows-based systems that do not have WOW64 functionality.

- **InstallShield Prerequisite Editor**—Use this tool to create new InstallShield prerequisites and modify existing ones.
- **Custom icons for Setup.exe and Update.exe**—Specify a custom icon (.exe, .dll, or .ico file) that you want to use for **Setup.exe** and **Update.exe** files that you create at build time. The icon is displayed on the Properties dialog box for **Setup.exe**; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. End users can also see the icon when they view your **Setup.exe** file in Windows Explorer.
- **Expiration dates for Setup.exe**—Set an expiration date, as well as an expiration message, for **Setup.exe**. If end users try to run **Setup.exe** on or after the date that you have specified in your project, the expiration message is displayed, and the installation exits.
- **Support for installation of multiple packages using transaction processing**—Windows Installer 4.5 and later include support for installing multiple packages using transaction processing. The Premier and Professional editions of InstallShield let you add chained .msi packages to an installation project. Your package, plus the added .msi packages, are chained together and processed as a single transaction. If one or more of the packages in the transaction cannot be installed successfully or if the end user cancels the installation, the Windows Installer initiates rollback for all packages to restore the system to its earlier state.
- **Ability to export and reuse various project elements**—Increase efficiency by moving pieces of an existing project (dialogs, custom actions, or features) to a merge module or another installation project.
- **Multiple-instance support**—Create an installation that lets end users install multiple instances of a product on the same machine and in the same user context.
- **Device driver support**—Device driver support in the Premier and Professional editions simplifies the process of installing device drivers from installation using the Driver Installation Frameworks for Applications (DIFxApp) from Microsoft.
- **Additional Dialog Themes**—Several dialog themes are available only in the Premier and Professional editions of InstallShield. The Limited and Express editions contain only two themes.
- **Conversion of Visual Studio merge module projects**—The Premier and Professional editions of InstallShield let you convert a Visual Studio merge module project to an InstallShield merge module project; this is necessary if you want to build a merge module for consumption in other projects.
- **COM+ application proxy support**—Manage COM+ application proxies during your installation. A COM+ application proxy consists of a subset of the attributes of the server application, and it enables remote access from a client machine to the machine where the application resides.

For additional details about the features that are included with each edition, contact InstallShield Sales, or visit http://www.installshield.com.

To learn about upgrading from one edition to another, see the appropriate topic:

- Upgrading from the Express Edition to the Professional or Premier Editions
- Upgrading from the Professional Edition to the Premier Edition

Upgrading from the Express Edition to the Professional or Premier Editions

If you have an Express project type that was created using the Express edition of InstallShield, that project is upgraded to a Basic MSI project when you open it in the Premier or Professional edition of InstallShield. This upgrade is done because Basic MSI is the comparable project type for the Express project type. QuickPatch projects created with the Express edition remain as QuickPatch projects in the Premier and Professional editions of InstallShield.

When you open any project created with the Express edition in the Premier or Professional edition, InstallShield converts the file from an .ise file to an .ism file. A backup of your .ise file is saved before the conversion takes place.

Billboard properties from Express edition projects will not be migrated since they are not supported in Basic MSI projects. You will receive warning 701 when the billboards property is specified in Express edition projects.



Task ***To upgrade an Express edition project:***

Open the project in the Premier or Professional edition of InstallShield. A dialog box opens to ask if you would like to upgrade your project.

- Click **Yes** to upgrade your project. Your project opens in InstallShield.
- Click **No** to leave the project unopened.

A backup copy of your original project is created and stored in the location specified in the dialog box.

Upgrading from the Professional Edition to the Premier Edition

When you open a Professional edition project in the Premier edition, your project does not need to be converted. It opens seamlessly in the Premier edition, enabling you to take advantage of Premier edition–only features. To learn about the specific features, see Upgrading from Other InstallShield Editions.

For additional information about the features included with each edition, visit http://www.installshield.com.

Converting or Importing Visual Studio Projects into InstallShield Projects

Visual Studio includes limited support for creating setup and merge module projects. InstallShield lets you convert or import these types of Visual Studio projects into InstallShield projects so that you can use the advanced features and functionality in InstallShield to create installations and merge modules.

With InstallShield, you can do the following:

- Import a Visual Studio setup project (.vdproj) into an InstallShield Basic MSI or Merge Module project (.ism), or import a Visual Studio merge module project (.vdproj) into an InstallShield Basic MSI or Merge Module project (.ism). These tasks enable you to create InstallShield installation and merge module projects that contain the

same data and settings that were in your Visual Studio project. During the import process, you can choose to import or ignore certain settings in the Visual Studio project.

- Convert a Visual Studio setup project to an InstallShield Basic MSI project, or convert a Visual Studio merge module project to an InstallShield merge module project. Converting a Visual Studio merge module project to an InstallShield Merge Module project is necessary if you want to build a merge module for consumption in other projects.



Note *• If your Visual Studio project contains one or more project outputs, you can use InstallShield to import that Visual Studio project into an InstallShield project; however, InstallShield cannot convert that Visual Studio project into an InstallShield project.*

The following versions of Visual Studio are supported:

- Visual Studio 2010
- Visual Studio 2008
- Visual Studio 2005
- Visual Studio .NET 2003
- Visual Studio .NET

Benefits of Using an InstallShield Project Instead of a Visual Studio Project

If you convert or import your Visual Studio project into an InstallShield project, you can use the features in InstallShield to fully customize your project.

Following is a list of a few of the tasks that you can perform in InstallShield projects but not in Visual Studio projects:

- Modify the layout of dialogs through a visual Dialog Editor.
- Manage components and features.
- Store custom actions in the **Binary** table of the .msi or .msm database. (With Visual Studio, all custom actions must be installed with the product.)
- Manage SQL-related tasks, such as connecting to a SQL server and running SQL scripts.
- Manage IIS Web sites, applications, virtual directories, application pools, and Web service extensions.
- Add billboards that are displayed during file transfer.
- Modify XML files or other text files on the target system at run time.
- Create shortcuts to pre-existing files on a target system.
- Manage COM+ applications.

Import Process

InstallShield lets you import a Visual Studio setup or merge module project into an InstallShield Basic MSI or Merge Module project.

Note *• If the Visual Studio setup or merge module project that you want to import into an InstallShield project contains one or more project outputs, the InstallShield project must be in the same Visual Studio solution that contains the Visual Studio setup or merge module project and all of its project dependencies.*

In order to import a Visual Studio project that contains project outputs, you must be using InstallShield from within Visual Studio. If your InstallShield project is open in InstallShield, but not from within Visual Studio, and you try to import a Visual Studio project that contains project outputs into the InstallShield project, an error occurs.

Task ***To import a Visual Studio project (.vdproj) into an InstallShield project (.ism):***

1. In InstallShield, create or open the Basic MSI or Merge Module project.
2. On the **Project** menu, click the **Visual Studio Deployment Project Wizard** button.
3. Complete the panels of the Visual Studio Deployment Project Wizard.

InstallShield imports the Visual Studio project into your open InstallShield project based on the settings that you configured in the wizard. As InstallShield imports the project, it displays the status of the project import in the Output window. The Output window shows each step of the conversion process, and it lists any conversion errors and warnings.

Conversion Process

If you use InstallShield to convert a Visual Studio setup project, InstallShield creates an InstallShield Basic MSI project (.ism).

If you use InstallShield to convert a Visual Studio merge module project, InstallShield creates an InstallShield Merge Module project (.ism).

Task ***To convert a Visual Studio project (.vdproj) to an InstallShield project (.ism):***

1. Open InstallShield.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.
3. In the **Files of type** box, select **Visual Studio Setup Projects (*.vdproj)**.
4. Browse to the location of the Visual Studio project that you want to open, and select the project file.
5. Click the **Open** button.

InstallShield creates an InstallShield project based on the settings in the Visual Studio project. InstallShield stores the .ism file in the same folder as the .vdproj file. As InstallShield creates the .ism file, it displays the status of the project conversion in the Output window. The Output window shows each step of the conversion process, and it lists any conversion errors and warnings.

Once the conversion process finishes successfully, the new InstallShield project is displayed in InstallShield.

Post-Import and Post-Conversion Tasks

Prerequisite Tasks

Visual Studio lets you add one or more predefined prerequisites to one or more configurations in a Visual Studio setup project. The import and conversion processes in InstallShield attempt to convert all of the prerequisites in all of the configurations to equivalent InstallShield prerequisites. If InstallShield does not include a corresponding InstallShield prerequisite, warning -9071 occurs, alerting you that the prerequisite could not be converted. To resolve this warning, consider creating an InstallShield prerequisite that installs the required redistributable, and add that InstallShield prerequisite to your project. For more information, see Defining InstallShield Prerequisites and Working with InstallShield Prerequisites that Are Included in Installation Projects.

User-Interface Tasks

The import and conversion processes do not incorporate the dialogs from a Visual Studio project into the InstallShield project. Once you have imported or converted your project, you can use the Dialogs view in InstallShield to configure settings for the dialogs in your project.

Language Tasks

If you imported a Visual Studio project into an InstallShield project and the following conditions exist, InstallShield replaces the existing string entry values in your project with default string entry values for the language of your Visual Studio project:

- You indicate in the Visual Studio Deployment Project Wizard that you would like to import the language of the Visual Studio project.
- The language of your Visual Studio project does not match the default language in your InstallShield project. (In Visual Studio, the Localization property indicates the project's language. In InstallShield, the Default Language setting in the String Editor view indicates the project's default language.)
- InstallShield has support for the language that is used in your Visual Studio project. (If you are using the Professional edition of InstallShield, you may not have support for the language that is used in your Visual Studio project.)

For example, if you indicate in the Visual Studio Deployment Project Wizard that you would like to import the language of the Visual Studio project, if the language of your InstallShield project is Spanish, if the language of your Visual Studio project is German, and if you are using the Premier edition of InstallShield, InstallShield replaces the Spanish run-time strings in your project with the default German translations. Thus, if you edit a string entry value by revising a setting such as the Publisher setting in the General Information view, and then you indicate in the wizard that you want to import the language of a Visual Studio project, InstallShield overwrites the value of the Publisher setting—as well as values for other settings—with the default German string entry values.

Therefore, if you change the project language while importing your Visual Studio project, review the settings in the General Information view and the String Editor view, and modify the string entry values if appropriate.

.NET Installer Class Tasks

If your Visual Studio project includes a .NET installer class custom action, InstallShield configures the .NET installer class information for the .NET assembly's component during the conversion process, instead of creating a .NET installer class custom action. InstallShield does not include support for the Condition property of a .NET installer class custom action in Visual Studio. Therefore, if your Visual Studio project contains a .NET installer class custom action that has a condition, you may want to use the Components view in InstallShield to create a condition for the component that contains the .NET assembly after you have converted your project.

Additional Tasks

You can also use the other views in InstallShield to make additional changes to your project.

Note • *Visual Studio lets you specify a directory path that contains multiple formatted properties, such as [ProgramFilesFolder][Manufacturer]\[ProductName], for the application folder. Visual Studio projects use a directory custom action to resolve the path at run time. However, InstallShield does not support this type of directory path. Therefore, InstallShield resolves the path during the conversion process and uses the* ***INSTALLDIR*** *property for the path.*

Obtaining Updates for InstallShield

If you have an Internet connection, you can use the Check for Updates feature in InstallShield to obtain the latest InstallShield prerequisites, merge modules, and objects; service packs; patches; and other updates for the version of InstallShield that you are using.

Task ***To check for updates:***

On the **Tools** menu, click **Check for Updates**.

InstallShield launches FlexNet Connect, which checks for updates. When an update is available, you can do the following:

- View the updates for your system.
- View descriptions about the updates.
- Download and install the updates that you select.

Run-Time Language Support in InstallShield

Important • *Language support varies, depending on the edition of InstallShield that you are using, the project type, and the language of the InstallShield interface (English or Japanese).*

Note that support for Arabic (Saudi Arabia) and Hebrew is available only in Basic MSI, Merge Module, and Suite/Advanced UI projects in the Premier edition.

The InstallShield Premier edition includes default run-time strings in 35 supported languages. When you add a supported language to a project, that language is made available in various language-related settings throughout InstallShield. In addition, InstallShield adds translated string entries for that language to your project. The string entries are for the default dialogs, messages, and other end-user interface elements. For a list of the supported languages, see Language Identifiers.

The InstallShield Premier edition also lets you add unsupported languages, beyond the built-in 35 languages, to Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects through the New Language Wizard. An unsupported language is one in which none of the default run-time strings are translated. When you add an unsupported language to a project, that language is made available in various language-related settings throughout

InstallShield. In addition, InstallShield uses the strings from your project's default language as placeholders for the strings in that newly added unsupported language; you can use the String Editor view to provide translated strings for the unsupported languages.

The English version of the InstallShield Professional edition lets you select one of 33 languages (any of the supported languages, except for Arabic or Hebrew) to use as the run-time language for all of the projects that you create. The English version of the InstallShield Professional edition includes default run-time strings for the one language that you have selected.

The Japanese version of the InstallShield Professional edition lets you select English plus one additional supported language; that is, if you are using the Japanese version of the InstallShield Professional edition, you can create projects with English run-time strings as well as projects with run-time strings from one other supported language. The Japanese version of the InstallShield Professional edition includes default run-time strings in English, plus the one additional supported language that you have selected.

If you want to modify the default run-time strings or you are adding custom dialogs, messages, and other end-user interface elements to your project, you can enter the custom run-time strings directly in your InstallShield project. As an alternative, you can export a language's string entries to a file, translate the string values in the file, and then import the translated file into your project. This support is available in all editions of InstallShield.

To learn more about language support in InstallShield, see the following sections of the documentation:

- Localizing the End-User Interface
- Creating Multilingual Installations
- Code Page Requirements for Language Support
- Settings for Languages

Code Page Requirements for Language Support

If you plan on creating releases that include run-time strings in a language that includes double-byte characters (for example, Japanese, Greek, or Korean), determine whether you need to install any code pages on your build machine.

Unicode encoding is used to display run-time strings throughout InstallShield. With Unicode encoding, double-byte characters are displayed correctly within InstallShield, even if InstallShield is on a system that does not have a project's target languages installed. In addition, you can build releases on a system that does not have the release's target languages.

However, if you use ANSI encoding for run-time strings (for example, if you export the string entries from a project as an ANSI text file and then import the translated ANSI text file into your project), you need to install the code pages for those languages on your build machine. The code pages allow your system to accurately represent the characters of those languages. If your build machine does not have the code pages installed, InstallShield reports a build error when you try to build a release, and the installation does not display double-byte characters properly at run time.



***Tip** • It is recommended that you use Unicode encoding instead of ANSI encoding when you are exporting string entries to a file for translation.*

Installing Supplemental Language Support on a Build Machine

The following instructions explain how to install code pages on a build machine that has Windows XP. This may be necessary if you want to build releases that contain Chinese, Japanese, and Korean languages. On later operating systems, the code pages are typically installed by default.



Task ***To install code pages on a build machine that has Windows XP:***

1. In the **Control Panel**, launch **Regional and Language Options**.
2. Click the **Languages** tab.
3. Click the **Details** button.
4. Click the **Add** button.
5. Select the language for which you want to add a code page.
6. Click **OK**.

The following instructions explain how to install files for right-to-left languages such as Arabic and Hebrew on a build machine that has Windows XP. On later operating systems, these files are typically installed by default. If these files are not installed, the String Editor view may display the strings left to right instead of right to left.



Task ***To install right-to-left language support on a build machine that has Windows XP:***

1. In the **Control Panel**, launch **Regional and Language Options**.
2. Click the **Languages** tab.
3. In the **Supplemental language support** area, select the **Install files for complex script and right-to-left languages** check box.
4. Click the **Add** button.
5. Select the language for which you want to add a code page.
6. Click **OK**.
7.

Supported Application Programming Languages

InstallShield allows you to create installations for applications created in *any* programming language—including applications created with Java, Pascal, C++, Visual Basic, Delphi, C# .NET, Visual Basic .NET, ASP .NET, and Cobol.

Even if you created your application using a language other than those listed above, you can use InstallShield to create an installation for your application. In addition—if you are not deploying an application, but want to package files or a database—you can use InstallShield to assist with those tasks.

3

Tutorials

This section contains tutorials that walk you through the process of creating an installation using InstallShield.

Table 3-1 • Available Tutorials

Tutorial	Description
InstallScript Project Tutorial	Leads you step-by-step through the process of creating an InstallScript installation project. The tutorial also provides information about the various tasks associated with installation projects and introduces you to the views in the InstallShield interface in which you can accomplish these tasks.
Basic MSI Project Tutorial	Teaches you how to a Basic MSI installation project. The tutorial also provides information about the various tasks associated with installation projects and introduces you to the views in the InstallShield interface in which you can accomplish these tasks.
Globalization Tutorial	Shows you how to add languages and language-specific components to your project. Additionally, it shows you how to conditionally install components based on the target system's language. ***Note*** • *The Premier edition of InstallShield must be installed on your development system in order to successfully complete this tutorial.*

InstallScript Project Tutorial

This tutorial guides you through the process of creating, building, running, and enhancing an InstallScript installation using InstallShield.

The tutorial is divided into several steps. After the first step—Step 1: Creating, Building, and Testing Projects—the other steps can be performed independently, and in any order, so you can focus on the information relevant to your work.

In this tutorial, you will learn how to handle many of the tasks that an installation needs to address, including:

- Installing files
- Setting up shortcuts and registry data
- Conditionally installing data
- Changing the installation's user interface
- Building release images
- Testing the installation

Throughout the tutorial are links to related topics in the Help Library.

Step 1: Creating, Building, and Testing Projects

This step demonstrates how to create an installation project, build a release image, and test the installation. After completing this step, you will know how to:

- Use the Project Assistant to create a new project.
- Specify global properties of your installation project.
- Define setup types, features, components, and file links.
- Build a release image for duplication and distribution.
- Run your installation from the InstallShield user interface.

An installation is made up of three levels:

Table 3-1 • Levels Within an Installation

Level	Description
Components	A component is the smallest separately installable piece of your product from the developer's viewpoint. A component specifies files, shortcuts, registry data, and other data to be installed on the target system. The end user never directly interacts with components. A component can be placed in more than one feature, and the component's data will be installed if the user selects at least one feature associated with the component.

Table 3-1 • Levels Within an Installation (cont.)

Level	Description
Features	A feature is the smallest separately installable piece of your product from the user's viewpoint. If the user selects the Custom setup type, a dialog is displayed in which the user can choose which features to install. Each feature contains components.
Setup Types	Setup types are predefined collections of features. By default, an installation offers Complete and Custom setup types, and the end user selects which setup type to install in the SetupType dialog.

The installation that you will create in this tutorial installs and configures an application called *Tutorial App*. The source files for Tutorial App are located in one of the Samples subfolders within the InstallShield Program Files folder. The default installation location is:

C:\Program Files\InstallShield\2016\Samples\WindowsInstaller\Tutorial Project

Creating a Project with the Project Assistant

After launching InstallShield, create a new InstallScript project by doing the following:

1. Open the **New Project** dialog box by doing one of the following:
   - Click the **Create a new project** link in the **Start Page's Project Tasks** section (on the left of the page).
   - On the **File** menu, click **New**.
   - Click the toolbar's **New Project** button
2. In the **New Project** dialog box, click the **InstallScript** tab.
3. In the **InstallScript tab's** list view, select the **InstallScript Project** icon.
4. In the **Project Name** edit box, type the project name *Tutorial*.
5. Click **OK**.

There are many other ways to create a project, such as updating a project created using InstallShield Professional. For more information, see Creating New Projects.

InstallShield creates a project file called *ProjectName*.ism, in this case Tutorial.ism. The project file stores all the settings you make in the InstallShield user interface. To move a project to another machine, copy the .ism file (and the installation source files) to the other system.



Tip • *You can change the directory where new project files are created by choosing Options from the Tools menu, selecting the File Locations tab, and entering the new location in the "Project Location" field.*

Your new project is opened to its Project Assistant tab. To begin using the Project Assistant, click the Next button in the lower right corner.

Tip • *You can do the Project Assistant's steps in any order, and switch at any time (by clicking the appropriate tab) between the Project Assistant and the Installation Designer, in which you can add more complex and powerful elements to your installation project.*

Specifying Application Information

The Application Information page lets you specify general information about the application that you are installing.

Task ***For this tutorial, do the following:***

1. In the **Specify your company name** edit box, type the name *Tutorial Co*.
2. In the **Specify your application name** edit box, type the name *Tutorial App*.
3. Leave the other fields unchanged.

The value you enter in the application name field is used in dialogs displayed to the end user, as the display name for your application in Add or Remove Programs in the Control Panel. The application name and company name you enter determine the default location of application shortcuts on the Windows Start menu, and the default value for the *TARGETDIR* system variable, which specifies the default destination for components' files.

Customizing the Installation Architecture

The Installation Architecture page lets you specify the features you want to be displayed by your installation. A feature is the smallest separately installable piece of your product from the end user's standpoint. Individual features are visible to end users when they select a Custom setup type.

Note • *Features can contain subfeatures, subsubfeatures, and so forth, to as many levels as your installation requires.*

Task ***For this tutorial, do the following:***

1. For the **Do you want to customize your Installation Architecture?** question, select **Yes**.
2. Select the existing **DefaultFeature** feature and rename it **ProgramFiles**.
3. Create a new feature called **HelpFiles**. To do so, click **Installation Architecture** and then click the **New** button.

Adding Files to Your Project

The Application Files page lets you specify the files you want to link with each of your features.

First, select from the list of features the feature in which you want to insert files. To add file links, click the Add Files button, and then browse for the file or files you want to include in the selected feature.

Task ***For this tutorial, add the Tutorial.exe file to the ProgramFiles feature by doing the following:***

1. In the list of features, click **ProgramFiles**.
2. In the tree control (with top node Destination Computer), select **Application Target Folder**.
3. Click **Add Files**.
4. Browse for Tutorial.exe, which is located in your source directory.
5. When the "The file you have added ... may have dependencies" message appears, click **No**; Tutorial.exe has no dependencies.

Creating Shortcuts

The Application Shortcuts page lets you specify shortcuts for your application's files on the target system's desktop or Start menu. By default, this page displays a shortcut for each executable that you have included in your installation project; you can delete these, and add shortcuts to other files that you have included in your installation project.

For this tutorial, leave this page's default specifications unchanged: a shortcut to Tutorial.exe on the Start menu.

Configuring Registry Data

The Application Registry page lets you specify any registry entries that your application requires.

Note • *An InstallScript project includes script code that by default creates the application uninstallation key (Software\Microsoft\Windows\CurrentVersion\Uninstall\<GUID> under the HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER root key as appropriate) and its values and data; you do not need to specify these registry entries.*

For this tutorial, do not specify any registry entries in this page. Registry entries are covered in

Step 2, Shortcuts and Registry Data, of the tutorial.

Selecting Dialogs with the Installation Interview

The Installation Interview page lets you specify the dialogs that you want the end user to see when your installation runs. Based on your answers to the questions on this page, the Project Assistant adds the corresponding dialog function to your installation script. Script changes related to dialogs are covered in Step 6 of the tutorial.

Task ***For this tutorial, do the following:***

1. Select the **No** option button under the text "Do you want to display a License Agreement Dialog?"
2. Leave the other option buttons set to **Yes**.

Choosing a Language for Your Installation

The Installation Localization page lets you specify the languages that your installation supports. It also lets you specify string values and associated identifiers, which you can use in your end user interface to make your installation more easily localizable in other languages.



***Edition** • Multilingual language support is available in the Premier edition of InstallShield.*



Task ***For this tutorial, change the display name of the HelpFiles feature by doing the following:***

1. In the list box, select **Feature String Data**. The table on the right displays all of the feature string entries.
2. In the **Value** column, click **HelpFiles** (the value associated with the identifier IDS_FEATURE_DISPLAY_NAME2), and change it to **Help Files**; that is, add a space.

Building Your Installation

The Build Installation page lets you specify what type of distribution you want to build (and, optionally, the location to copy the distribution files to).



Task ***For this tutorial, do the following:***

1. Select the **CD-ROM** option.
2. Click **Build Installations**.

The output window opens with the Build tab uppermost and displays information about the progress of the build. The build is finished when the Build tab displays the line "Build finished at *date and time*".

Running Your Installation

To run your installation from within InstallShield, click the Run toolbar button or press CTRL+F5.

The installation displays the dialogs that you specified in the Installation Interview Page of the Project Assistant. The values you entered in the Project Assistant are displayed to the end user in the appropriate dialogs. For example, at run time, the default value of **TARGETDIR** that you specified in the Project Assistant appears in the Choose Destination Location dialog. If the end user browses for a different destination directory, **TARGETDIR** stores the new value.

After the installation is complete, you can browse for the directory and find the files installed by your setup program. If the installation was successful, you will see the tutorial files installed.

Maintenance Mode

When a user runs an installation a second (or later) time for a product installed on their system, the installation runs in *maintenance mode*. Maintenance mode allows the user to modify feature selections from the first-time installation, repair the features already installed, or remove the entire program.

Uninstalling the Program

To uninstall the program, click the Run button (or press CTRL+F5), and select Remove from the Setup Maintenance dialog. This is the same behavior a user sees when selecting your application in Add or Remove Programs.

Now that you have created a basic installation project, click the Installation Designer tab (near the top of the window) to expand and fine-tune your installation as illustrated in the next step of the tutorial.

Working with the InstallShield Interface

Now that you have created a basic installation project, click the Installation Designer tab (near the top of the window) to expand and fine-tune your project in the InstallShield user interface. The InstallShield user interface is arranged in functional categories that help you add or edit information in your project. This and later steps of the tutorial explore several of the different InstallShield views.

After completing this step, you will know how to:

- Set display properties for your program features.
- Define your program's setup types.
- Create components and add file links.

Setting Feature Properties

First, you will set additional properties for the features you created in the Project Assistant, such as the feature display name and description. To edit the feature properties, go to the Features view in the IDE.



Task ***To set feature properties:***

1. On the **Go** menu, point to **Organization** and click **Features**.
2. In the **Features** explorer, click the DefaultFeature feature and set its **Description** property to **This feature contains the Tutorial App program files**.
3. Select the New Feature feature and set its **Description** property to **This feature contains the Tutorial App help files**. As you enter each description, InstallShield creates a string entry—displayed as {ID_STRING*n*}—to represent each value.
4. Rename the features in the **Features** view to have the same names as their respective display names. To rename a feature, click the feature twice to highlight its name, then type the new name.

At run time, if the end user chooses the Custom setup type, the installation displays a dialog that prompts the user to select which features to install. This dialog displays features using the display name and description you specified here.

Setting Setup Type Properties

Setup types are collections of features to be installed. A typical installation offers Complete and Custom setup types—where the Complete setup type installs all features, and a Custom setup type displays a dialog that lets the end user choose which features to install.

You modify setup type properties in the Setup Types view of the IDE (under the View List's Organization node).

Task ***For each setup type, select the features to be installed by selecting the boxes in front of the features' names:***

1. For the Complete setup type, select both features.
2. For the Custom setup type, select both features.

Creating Components and File Links

You can add links to additional files in the Files and Folders view. In this step, you will add a file to your HelpFiles feature. As you add files in the Files and Folders view, the IDE creates components for you according to Setup Best Practices rules.

Task ***To add the source file Tutorial.html to a new component in the Help Files feature:***

1. Go to the **Files and Folders** view (under the View List's **Application Data** node).
2. Select **Help Files** from the feature list at the top of the view.
3. In the "Destination computer's folders" area, right-click the **Destination Compute**r icon and verify that Show Components is selected.
4. Right-click the **Application Target Folder** icon and select **New Component**.
5. Rename the new component HelpComponent.
6. In the "Source computer's folders" area, browse for the source folder containing TutorialHelp.html.
7. Drag the TutorialHelp.html icon from the "Source computer's files" area and drop it on the **HelpComponent** icon.

This type of file linking, where the list of files linked to a component does not change, is called *static file linking*. To link to a directory (and possibly its subdirectories) whose contents might change between builds, see Dynamic File Linking.

Tip • *You can use the InstallShield dependency scanners to determine additional files required by your application that are currently not included in your project. For example, Tutorial App uses MFC, so it will be necessary to add the MFC Runtime object to your project in the Redistributables view if you intend to target systems that do not have the MFC run-time libraries installed.*

The next step of the tutorial explains how to build a release image for your installation project.

Building a Release

Before testing an installation, it is necessary to build a release. A release image contains all of the files to be distributed to the end user on a CD-ROM or floppy disk or from a network location.

The simplest way to build a new release is to use the Release Wizard. The Release Wizard is where you specify release properties, such as the type of media (CD-ROM, for example) to use and whether to compress files on the media. You can launch the Release Wizard by clicking the Release Wizard toolbar button, or by choosing Release Wizard from the Build menu.

Click Next in the Welcome panel to specify your release settings. You can click Help in any panel for more information about the current step.

Naming the Release

In the Specify a Release panel, specify a release name. The release name is used as the name of a folder in which your built release will be placed. For this example, create a new release called *cdrom*.

Selecting the Media Type and General Options

Media Type Panel

In the Media Type panel, you specify the type of media for which you are building a release. The media type you specify indicates the size of the disk image folders the Release Wizard creates: when you build a release for CD-ROM, the Release Wizard divides your disk images into folders, each of which is smaller than 650 MB.

For this tutorial, select CD-ROM.

Click Next to specify general options for your release.

General Options Panel

The General Options panel lets you do the following:

- Create a self-extracting executable file for distributing your installation
- Pass command line options to Setup.exe
- Pass preprocessor variable definitions to the compiler
- Select whether to place the compiled script file (.inx file) in a cabinet file

For this tutorial, leave the panel's default settings unchanged.

Specifying a Password and Supported Platforms

Password Panel

In the Password panel, you can specify a password for your installation, and, if you do, whether to execute the password-checking code in the OnCheckMediaPassword event handler function's default code.

For this tutorial, do not specify a password.

Click Next to specify the operating systems you want to support in the current release.

Platforms Panel

In the Platforms panel, you can specify the operating systems you want to support in the current release.

For this tutorial, do not change the default selection: **Use platforms specified by the Platforms project property**.

Specifying Setup Languages and Including Features

Setup Languages Panel

In the Setup Languages panel, you can specify the languages you want your installation to be able to run in, and whether to display a dialog that allows the end user to select the language in which they want the setup displayed.

The wizard will build into your installation only those language-specific elements, including string entries and dialogs, that you select in this panel. All language-independent resources, such as product properties and built-in actions, are included as a matter of course.

For this tutorial, leave the default selections unchanged.

Click Next to specify which features you want to include in the current release.

Features Panel

In the Features panel, you can specify which features are included in the built release.

For this tutorial, do not change the default selection: **Use the 'Include in Build' feature property to determine inclusion**.

Defining Media Layout and Dialog Appearance

Media Layout Panel

In the Media Layout panel, specify, for individual features or for all features, whether the features' files are stored in cabinet files or placed uncompressed in the disk image.

For this tutorial, do not change the default selection: "Cabinet File(s)".

User Interface Panel

In the User Interface panel, specify the look and feel of your installation's end user dialogs.

For this tutorial, leave the default selections unchanged.

Specifying Internet Options and Digitally Signing Your Application

Internet Options Panel

In the Internet Options panel, specify various Internet-related options. Any release can be run over the Internet, regardless of its media type.

For this tutorial, check the Create a default Web page for the setup box and leave the other default selections unchanged.

Digital Signature Panel

In the Digital Signature panel, you can digitally sign your application. Digitally signing your application helps to assure your end users that the code within your application has not been modified or corrupted since publication.

For this tutorial, leave the default selections unchanged.

Specifying Update and Postbuild Information

Update Panel

The Update panel lets you specify the release format and the existing releases for which the current release can be run as an update.

For this tutorial, leave the default selections unchanged.

Postbuild Options Panel

The Postbuild Options panel lets you copy disk image folders to a folder or FTP site, or execute a batch file, after the release build is complete.

For this tutorial, leave the default selections unchanged.

Reviewing Your Settings

The Summary panel displays the Release Wizard settings for the current release. If the settings are correct, select the Build the Release check box and click Finish to build your release. If not, click Back to modify the settings.

Status messages for the build in progress are displayed in the output window.

When the build is complete, the files to copy onto the CD are placed in the following directory:

<ProjectFolder>\Tutorial\cdrom\DiskImages\DISK1.

After making changes to your project in later steps of the tutorial, you can rebuild the latest release by clicking the Build toolbar button, choosing Build from the Build menu, or pressing F7.

You can now run the installation as you did previously in this tutorial. To run the installation as an Internet installation, click the toolbar's Open Release Folder button and launch Setup.htm.

The next step of the tutorial explains how to create shortcuts and registry data for an installation.

Troubleshooting Your Installation

After running your installation, if files are not installed, check the following parts of your project:

- Verify that TARGETDIR is set to the proper value. This is set in the General Information view.

  For this tutorial, the recommended value is as follows:

  [ProgramFilesFolder]TutorialCo\TutorialApp

- Verify that your setup types have features associated with them.
- Verify that your features have components and files associated with them.
- After making any changes to your installation, it is necessary to rebuild your project by clicking the Build button or pressing F7.

Step 2: Shortcuts and Registry Data

This step explains how to use the IDE to:

- Create program shortcuts
- Create registry information

Creating Shortcuts

You create and modify shortcuts in the Shortcuts view. The properties of a shortcut include its display name, its target executable and arguments, and the icon it displays. Using the Project Assistant, you have already created a shortcut to Tutorial App in the end user's Programs folder, under the Start menu.

Task ***In this step you create a shortcut on the end user's desktop.***

1. Go to the **Shortcuts** view.
2. Right-click the **Desktop** icon and select **New Shortcut**. The **Browse for Shortcut Target** dialog box opens.
3. In the dialog box, select **Application Target Folder** from the **Look in** drop-down menu and select Tutorial.exe from the files list.
4. Click **Open** to close the **Browse for Shortcut Target** dialog box.
5. Rename the shortcut icon to an internal name such as *tutorial*.

Next, set the following properties for the shortcut:

Table 3-2 • Shortcut Properties

Property	Value	Comment
Display Name	Tutorial App	InstallShield adds the display name to the project's strings (and displays the string identifier in curly braces in the Display Name field) so that the name can be easily localized to other languages.
Target	<TARGETDIR>\Tutorial.exe	
Icon File	<TARGETDIR>\Tutorial.exe	
Icon Index	0	
Working Directory	<TARGETDIR>	

Tip • *To create a shortcut to a file already located on the user's machine, enter the path to the file—using system variables to represent the path to the file, when possible. For example, to launch a copy of Windows Notepad located in the user's Windows or WinNT folder, enter the shortcut target as <WINDIR>\Notepad.exe.*

Creating Registry Data

Another common requirement for installations is to write information to the target system's registry. To add registry data to a component, you can use the Registry view.

For example, to create a registry value called *TutorialData* under HKEY_LOCAL_MACHINE\Software\Tutorial Co\Tutorial\1.00.0000:



Task ***To create registry data:***

1. Go to the **Registry** view.
2. In the **Destination computer's Registry view area**, right-click **Destination Computer** and select **New Registry Set**.
3. Rename the registry set *tutorial*.
4. Under the *tutorial* registry set, right-click HKEY_LOCAL_MACHINE and from the **New** submenu select **Key**.
5. Rename the key *Software*.
6. Repeat the process for subkeys named *Tutorial Co*, *Tutorial*, and *1.00.0000*.
7. In the **Destination computer's Registry data area**, right-click and select **New String Value**.
8. Rename the value *TutorialData*.
9. Double-click the TutorialData value and enter <*TARGETDIR*> in the **Value data** field.
10. Click the *tutorial* registry set, and in the **Registry Set Install Conditions** pane check DefaultComponent.

At run time, if the end user selects a setup type or collection of features that includes the Tutorial.exe component, the registry data is created on the target system.

Verifying that the Shortcut Was Created



Task ***To verify that your installation created the shortcut:***

1. Rebuild your project by clicking the **Build** toolbar button or pressing **F7**.
2. Run the project by clicking the **Run** button or pressing CTRL+F5 (first removing any existing version of the program from your system). A shortcut to Tutorial App should be present in the **Programs** folder of your **Start** menu.

Verifying that the Registry Data Was Created



Task ***To verify that the installation created the registry data:***

1. Launch Tutorial App from its shortcut.
2. From the **Tutorial** menu, choose **Verify Registry Data**. If the registry data was created, a message box displaying the text *<TARGETDIR>* is displayed.

The next step of the tutorial explains how to register a COM server (self-registering file).

Step 3: Registering COM Servers

For many files, the installation's only requirement is to copy the files from the source media to the target system. For others, the installer also needs to register the files with the target system. One category of file that needs extra handling is a *self-registering file*.

In this step you will create a component that installs and registers Tutorial.ocx and an HTML file that uses it.

A COM server is usually a DLL or .ocx file that requires extra information to be written to the target system's registry before applications and Web pages that use the self-registering file can find it.



Task ***To install a self-registering file:***

1. Go to the **Files and Folders** view.
2. At the top of the **Files and Folders** view, in the **View Filter** list, select the **ProgramFiles** feature.
3. In the **Source computer's folder pane**, browse for Tutorial.ocx in your source directory.
4. Drag Tutorial.ocx from the **Source computer's folders pane** and drop it into the **Destination computer's folders pane's Application Target Folder**. The component SelfRegFiles is created under Application Target Folder; this component contains Tutorial.ocx and has its Self-Register property set to **Yes**, as you can verify by right-clicking the component and selecting **Properties**.

Next, add the HTML file to a new component also associated with the ProgramFiles feature.



Task ***To add the HTML file to a new component associated with the ProgramFiles feature:***

1. In the **Destination computer's folders pane** of the **Files and Folders** view, right-click **Application Target Folder** and select **New Component**.
2. Rename the component *OcxHTML*.
3. Drag the file TutorialCtrl.html from the **Source computer's files view** into the *OcxHTML* component.



Task ***After rebuilding your release (by pressing F7) and running the installation (by pressing CTRL+F5), you can verify that the file was registered properly:***

1. Launch Tutorial App using its shortcut in the **Programs** menu, or by double-clicking its icon.
2. Choose **COM Server Test** from the **Tutorial** menu.
3. If the COM server was registered correctly, the HTML page displays a "success" message.

The next step of the tutorial demonstrates how to install files conditionally.

Step 4: Conditions and Properties

In this step, you will learn how to conditionally install data on a target system.

A common requirement for installations is to install certain files on a system only if particular conditions are met. For example, files may be specific to an operating system or language, or should be installed only if the user has appropriate privileges.

To install a component (and its files and other data) only on particular operating systems, you can use the component's Operating Systems property. You can modify a component's properties by opening the Setup Design view, expanding the feature icon that contains the feature, and selecting the desired component.



Task ***To create a component that will be installed only on systems that have Windows 8:***

1. Go to the **Setup Design** view.
2. Right-click the **HelpFiles** feature and select **New Component**.
3. Rename the component to *windows_8_files*.
4. Click the **Files** icon for the component and add the file ReadmeNT.txt from your source folder by right-clicking in the **Files** pane and browsing to the file.
5. Click the windows_8_files component to display the component's property grid.
6. Select the component's Operating Systems setting and click the browse button. The **Platforms** dialog opens.
7. Select the Windows 8 check box.
8. Click **OK**.

After you rebuild (by pressing F7) and run the installation (by pressing CTRL+F5), and any files or other data contained in the component will be installed only if the target system is running Windows 8.

The next step of the tutorial describes how to modify your project's script.

Step 5: Working with Scripts

InstallShield uses the InstallScript language to drive an installation. You can modify the project's script in the InstallScript view.



Tip • *Press CTRL+M to maximize the Script Editor; press CTRL+M again to restore it.*

InstallScript project scripts use an event-driven model, where a series of predefined functions are called in a specific order. For information about the built-in categories of event handlers and the order in which event handler functions are called, see Event Handlers.

The functions already defined in your script appear in the Functions tree. To view or edit an existing function, click its name in the Functions tree.

***Note** • Event handler functions are called even if they do not explicitly appear in your script. If an event handler function does not appear in your script, its default code is used.*

To add and edit an event handler function, select the desired event category (such as Before Move Data) from the script editor's left drop-down list, and then select the name of the desired event (such as Begin) from the right drop-down list. Event handler functions that are already explicitly defined in your script appear in boldface text.

For example, the OnBegin event handler is the first function called in an installation script, for both a first-time installation and maintenance mode.

Task ***To add your own code to the OnBegin event handler, begin by creating the function:***

1. Select **Before Move Data** from the event category list on the left.
2. Select **Begin** from the event handler list on the right.

The following code is added to your script:

```
////////////////////////////////////////////////////////////////////////////
//
//  FUNCTION: OnBegin
//
//  EVENT: Begin event is always sent as the first event during
//  installation.
//
////////////////////////////////////////////////////////////////////////////
function OnBegin( )
begin
    // TO DO: you may change default non-UI settings, for example
    //
    // You may also perform your custom initialization steps, check requirements, etc.
end;
```

You can place any code you want to execute at the beginning of your installation in the **OnBegin** function.

Using the Function Wizard

This example demonstrates how to use the Function Wizard to add a call to the MessageBox function. This displays a message when the user begins the installation.

Task ***To add the MessageBox function to your OnBegin function:***

1. Delete the comments (lines beginning with `//`) between the lines reading begin and end;, and place the text insertion point between the lines.
2. Press CTRL+I to launch the **Function Wizard**.
3. In the **Function Category** list, select **Built-in dialog**.
4. In the **Function Name** list, select **MessageBox**.
5. Click **Next**.

6. In the **szMsg** field—which contains the message you want to display—type "*Welcome to the Tutorial installation!*" (including the quotation marks).
7. In the **nType** drop-down list—which specifies the type of message box to display—select **INFORMATION**.
8. Click **Finish** to paste your function call into the script.

Your **OnBegin** function should now appear as follows:

```
function OnBegin( )
begin
MessageBox ( "Welcome to the Tutorial installation!" , INFORMATION );
end;
```

Compiling the Script

After modifying the script, you must compile the script for the changes to take effect. Compile the script by clicking the Compile toolbar button, choosing Compile from the Build menu, or pressing CTRL+F7. Status messages are displayed during compilation in the output window of the IDE.



Tip *• If compiling the script results in any errors or warnings, you can double-click the error message to highlight the script line where the error occurred.*

When you run the installation (by clicking the Run toolbar button, or by pressing CTRL+F5), the message box appears before the Welcome dialog is displayed.

Displaying the Message only for First-Time Installation

The **OnBegin** function is called for both a first-time installation and a maintenance-mode installation, which means the message box will be displayed when the user reinstalls or removes the program. To display the message box only for a first-time installation, you can place your MessageBox call inside an if statement that determines if the user is running the installation for the first time.

InstallScript defines a system variable called *MAINTENANCE*, which is true if the installation is running in maintenance mode and false otherwise. To display your message box only for a first-time installation, change the **OnBegin** function to read as follows:

```
function OnBegin( )
begin
if (!MAINTENANCE) then
    MessageBox("Welcome to the Tutorial installation!", INFORMATION);
endif;
end;
```

After you compile and run the installation, the message box appears only for a first-time installation, and not for maintenance mode.

InstallScript

The InstallScript language contains hundreds of functions for performing installation-related tasks, such as working with the registry and INI files, testing characteristics of the target operating system, and displaying dialogs. For details and examples, see the InstallScript Language Reference.

The next step of the tutorial explains how to modify the dialogs that are displayed by your installation.

Step 6: Changing the User Interface

This step describes three ways you can modify the user interface of your installation:

1. Taking different actions based on user input.
2. Modifying user-interface-related script functions to display different dialogs.
3. Modifying the layout and properties of a dialog using the **Dialog Editor**.

Handling User Input

In your InstallScript code, the event handler functions that contain the dialog functions displayed at run time are:

- **OnFirstUIBefore**, which contains the dialogs to be displayed before data transfer for a first-time installation.
- **OnFirstUIAfter**, which contains the dialogs to be displayed after data transfer for a first-time installation.
- **OnMaintUIBefore**, which contains the dialogs to be displayed before data transfer for a maintenance-mode installation.
- **OnMaintUIAfter**, which contains the dialogs to be displayed after data transfer for a maintenance-mode installation.

Note • *The OnMaintUIBefore and OnMaintUIAfter event handler functions are not called if the project's Maintenance Experience property is set to "No uninstall or maintenance".*

A default InstallScript project created with the Project Assistant defines the **OnFirstUIBefore** event handler function, which defines the user interface for a first-time installation. **OnFirstUIBefore** calls dialog functions to display the dialogs that you specified in the Project Assistant's Installation Interview page. For example, the following code displays a dialog that prompts the end user to enter a user name and company name:

```
szMsg = "";
szTitle = "";
nResult = SdRegisterUser( szTitle, szMsg, szName, szCompany );
```

The end user's user name and company name are returned in the last two variables, which you can then use in any way you want, for example, to create a registry key or check against information you have stored in a file.

Changing the Dialogs Displayed

The InstallScript language includes many predefined dialogs that you can display in your installation's user interface.

Task ***To replace the default user information dialog with one that also prompts the user for a serial number, you can replace the SdRegisterUser function with a call to SdRegisterUserEx:***

1. In the **InstallScript** view, select **OnFirstUIBefore** from the **Functions** node.
2. Declare the string script variable szSerial by adding the following to the declarations before begin in the **OnFirstUIBefore()** function:

   ```
   string szSerial;
   ```

3. Delete the lines containing the call to SdCustomerInformation:

```
Dlg_SdRegisterUser:
        szMsg = "";
        szTitle = "";
        nResult = SdRegisterUser( szTitle, szMsg, szName, szCompany );
        if (nResult = BACK) goto Dlg_SdLicense2;
```

4. Replace the deleted lines with the following:

```
Dlg_SdRegisterUserEx:
        szMsg = "";
        szTitle = "";
        szSerial = "";
        nResult = SdRegisterUserEx(szTitle, szMsg, szName, szCompany, szSerial );
        if (nResult = BACK) goto Dlg_SdLicense2;
```

5. Change goto statements that previously pointed to Dlg_SdRegisterUser to point to Dlg_SdRegisterUserEx.
6. Press CTRL+F7 to compile the script.

Using this approach, you can insert additional dialog in your installation's user interface, or replace existing ones.

Using the Dialog Editor

The Dialog Editor allows you to modify the appearance of dialog displayed by your installation.

As shown in the previous step, the serial number entered by the end user is displayed in the SdRegisterUserEx dialog as plain text.

Task ***To modify the dialog so the password is hidden as the end user types it:***

1. In the **Dialogs** view (which is under the List View's User Interface node), right-click the **SdRegisterUserEx** icon and select **Edit**.
2. Select the **English (United States)** icon.
3. In the **Dialog Editor**, select the edit field under the **Serial Number** label.
4. Change the **Password** property of the **Edit** control from False to True.

Tip *• To maximize the Dialog Editor view, press CTRL+M.*

After rebuilding the project (by pressing F7) and running it (by pressing CTRL+F5), the serial number entered by the user will be hidden.

Tip *• To restore a dialog to its default appearance, right-click the dialog's icon and select Revert Dialog to Default.*

Basic MSI Project Tutorial

This tutorial guides you through the process of creating, building, running, and enhancing a Basic MSI installation project using InstallShield.

The tutorial is divided into several steps. After the first step—Step 1: Creating, Building, and Testing Your Project—the other steps can be performed independently, and in any order, so you can focus on the information relevant to your work.

In this tutorial, you will learn how to handle many of the tasks that an installation needs to address, including:

- Installing files
- Setting up shortcuts and registry data
- Conditionally installing data
- Registering COM servers
- Changing the installation's user interface
- Building releases
- Testing the installation

Throughout the tutorial are links to related topics in the InstallShield Help Library.

Step 1: Creating, Building, and Testing Your Project

This step demonstrates how to create an installation project, build a release image, and test the installation. After completing this step, you will know how to:

- Use the Project Assistant to create a new project.
- Configure many of the settings in your installation project.
- Define features, components, and file links.
- Build a release for duplication and distribution.
- Run your installation.

Basic MSI Installation Building Blocks

A Basic MSI installation is made up of two levels:

Table 3-1 • Levels Within a Basic MSI Installation

Level	Description
Components	A component is the smallest separately installable piece of your product from the developer's viewpoint. A component contains files, shortcuts, registry data, and other data to be installed on the target system. The end user never directly interacts with components. A component can be placed in more than one feature, and the component's data will be installed if the user selects at least one feature associated with the component.
Features	A feature is the smallest separately installable piece of your product, from the end user's viewpoint. If the end user selects the Custom setup type during the installation, a dialog with which the user can choose which features to install is displayed. Each feature contains components.

Tutorial Files

The installation that you will create in this tutorial installs and configures an application called *Tutorial App*. The source files for Tutorial App are located in one of the Samples subfolders within the InstallShield Program Files folder. The default installation location is:

C:\Program Files\InstallShield\2016\Samples\WindowsInstaller\Tutorial Project

Creating a New Basic MSI Project

The first step in the tutorial is to create a new Basic MSI project.



Task ***To create a new Basic MSI project:***

1. On the **File** menu, click **New**, or click the **New Project** button in the toolbar. The **New Project** dialog box opens.
2. Click the **Windows Installer** tab and select the **Basic MSI** project type.
3. In the **Project Name** field, enter **Tutorial**.
4. In the **Location** setting, leave the default value.
5. Select the **Create the project in 'Project Name' subfolder** check box.
6. Click **OK**.

InstallShield creates and opens your project.

InstallShield creates a project file called *ProjectName*.ism, or in this case, **Tutorial.ism**. The project file stores all of the settings that you configure in InstallShield. To move a project to another machine, copy the .ism file (and the installation source files) to the other system.

Tip • *To change the default directory where new project files are created, open the Options dialog box (available when you click Options on the Tools menu). On the File Locations tab, in the Location setting, enter a new path.*

Specifying Application Information

After you create a new project, the Project Assistant launches to help you specify project and application information. The first page in the Project Assistant provides a graphical overview of the installation creation process. To begin using the Project Assistant, click the Application Information icon at the bottom of the view.

The Application Information page is where you specify general information about the application that your project will install.

Task ***To specify application information for the tutorial:***

1. In the **Specify your company name** setting, enter **TutorialCo**. This automatically updates the information in the **Specify your company Web address** setting.
2. In the **Specify your application name** setting, enter **TutorialApp**. The value that you enter is used on dialogs that are displayed to end users. It is also used as the display name for your application in Add or Remove Programs.
3. In the **Specify your application version** and **Specify your company Web address** settings, leave the default values.
4. Click the **Browse** button under the **Application Icon** setting, and browse to the **Tutorial.exe** location. The default location is **C:\Program Files\InstallShield\2016\Samples\WindowsInstaller\Tutorial Project**. Open the .exe file and select the **Icon Index:0**.

The Application Information panel will look like the following when you are finished.



Figure 3-1: Application Information Panel

The application name and company name you enter determine the default location of application shortcuts on the *Windows Start* menu, and the default value for the *INSTALLDIR* property, which specifies the default destination for your program's files.



***Note** • The default value of INSTALLDIR is [ProgramFilesFolder]Your Company Name\Your Product Name. The special form [ProgramFilesFolder] expands to the location of the user's Program Files folder at run time. For a list of the other directory properties that are defined by Windows Installer, see the System Folders Set by the Installer section in Windows Installer Property Reference.*

Setting Installation Requirements

The Installation Requirements page allows you to easily set installation requirements for the target system. For example, if your application requires a specific operating system in order to run properly, you can indicate that in the first section of this panel.

Operating System Requirements

If your application required a certain version of Windows or later to run on the target system, you would select Yes and then select the operating systems with which your application can run properly.

For the tutorial, leave *No* selected

Software Requirements

If your application's installation requires that a particular piece of software be installed on the target system, select *Yes* and select the required software. To customize the run-time message that will be displayed if the required software is not present on the target system, click on the run-time message and edit.



***Note** • The run-time message is not displayed in this section until a software requirement is selected.*

For the tutorial, leave *No* selected.

Customizing Installation Architecture

The Installation Architecture page lets you specify the features that you want your installation to display to end users. A feature is the smallest separately installable piece of your product from the end user's standpoint. Individual features are visible to end users when they select a Custom setup type during installation.



***Note** • Features can contain subfeatures, subsubfeatures, and so forth, to as many levels as your installation requires.*

Your installation architecture currently contains a default feature, *Tutorial_Files*. The default feature is always installed when an end user runs your installation. In this step, you will add another feature to the installation architecture.



Task ***For the tutorial, add a new Help_Files feature:***

1. Select **Yes** for the **Do you want to customize your Installation Architecture?** option.
2. Right-click the **Installation Architecture** node and select **New**.
3. Name the new feature *Help_Files*.

When you finish this step, your installation architecture will look like this:

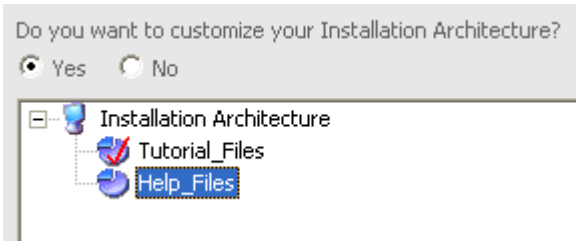


Figure 3-2: Installation Architecture

Adding Files to Your Project

The next step is to add your application's files to the installation project. The Application Files page lets you specify the files you want to associate with each of your features.

In this step, you will add the Tutorial executable file to the Tutorial_Files feature.

Task ***To add the Tutorial executable to the Tutorial_Files feature:***

1. Select the Tutorial_Files feature from the drop-down list of features at the top of the page.
2. In the tree control (with top node Destination Computer), select the **INSTALLDIR** node.
3. Click **Add Files**. An **Open** dialog box opens.
4. Browse to Tutorial.exe, which is located in the Tutorial Files source directory.
5. Click **Open** to add the file to the Tutorial_Files feature.
6. When the **file you have added ... may have dependencies** message appears, click **No**. Tutorial.exe has no dependencies. The file is added to the feature and appears in the file list panel on the right.

Note *• The key icon next to the file indicates that this file is the key file of the feature's component. Windows Installer requires that most components have a single key file. The Windows Installer service uses a component's key file for several purposes, including checking for the file's existence to determine if a component needs to be repaired and using the key file as the default target for a shortcut. When you add an executable file to a feature in a Basic MSI project, the Project Assistant automatically sets it as the key file of the component it creates behind the scenes for the file. For more information, see Setting Component Key Files.*

Creating Shortcuts

The Application Shortcuts page lets you specify shortcuts for your application's files on the target system's desktop or Start menu. By default, this page displays a shortcut for each executable that you have included in your installation project. You can delete these, and add shortcuts to other files that you have included in your installation project.



Task ***To activate the shortcut to Tutorial.exe:***

Click the Launch Tutorial.exe icon. Leave the default setting, **Create shortcut in Start menu**, selected. InstallShield will create a shortcut to Tutorial.exe on the end user's **Start** menu when the installation is run.

Configuring Registry Data

The Application Registry page lets you specify any registry entries that your application requires.

For the tutorial, do not specify any registry entries in this page. Registry entries are covered in Step 2: Shortcuts and Registry Data.

Selecting Dialogs with the Installation Interview

The Installation Interview page lets you specify the dialogs that you want end users to see when your installation runs. Based on your answers to the questions on this page, the Project Assistant adds the corresponding dialog to your installation project.



Task ***To specify dialogs for the tutorial, do the following:***

1. **Do you want to display a License Agreement Dialog?**—Select **No**. If you selected **Yes** for this option, you would be able to browse to your license agreement file.
2. **Do you want to prompt users to enter their company name and user name?—**Select **Yes**. The installation displays a dialog requesting this information.
3. **Do you want your users to be prompted to modify the installation location of your application?—**Select **Yes**. For more information, see Allowing End Users to Modify the Installation Location.
4. **Do you want users to be able to selectively install only certain parts of your application?—**Select **Yes**. For more information, see Creating Selectively Installable Installations.
5. **Do you want to give users the option to launch your application when the installation completes?—** Select **Yes** and browse to the Tutorial.exe file (located in [ProgramFilesFolder]TutorialCo\Tutorial). When this option is set to **Yes**, the final dialog in the installation presents a check box that allows the end user to immediately launch your application upon clicking the **Finish** button.

Choosing a Language for Your Installation

The Installation Localization page lets you specify the languages your installation supports, and specify string values and associated identifiers, which you can use in your end user interface to make your installation more easily localizable in other languages.



***Edition** • For language options in addition to the language that you chose when you installed InstallShield, you must have the Premier edition of InstallShield.*

Task ***For this tutorial, leave English (United States) selected and change the display names of the installation's features by doing the following:***

1. In the list box, select **Feature String Data**. The table on the right displays all of the feature string entries.
2. In the **Value** column, click **Tutorial_Files** (the value associated with the identifier IDS_FEATURE_DISPLAY_NAME2) and change it to **Tutorial Files**.
3. Click **Help_Files** (the value associated with the identifier IDS_FEATURE_DISPLAY_NAME3) and change it to **Help Files**.

Note • *For more information, see Creating Multilingual Installations.*

Building Your Installation

After defining your installation project's architecture, adding your application files, creating shortcuts, and selecting dialogs, you are ready to build the installation.

The Build Installation page lets you specify what type of distribution you want to build and, optionally, the location to which you want to copy the distribution files.

Task ***To build the installation you have just created:***

1. Select the CD-ROM option.
2. Click **Build Installations**.

The Output window opens and displays information about the progress of the build.

Note • *The build generates warning -7235. This is expected. You can continue to the next step without resolving this warning, or to resolve this warning, configure the settings in the Software Identification Tag area of the General Information view as needed. If Yes is selected for the Use Software Identification Tag setting but you have not entered values in one or more of the required identification settings (the Unique ID, Tag Creator, and Tag Creator ID settings in the General Information view), build warning -7235 occurs, once for each of the settings that are blank. For more information, refer to Including a Software Identification Tag for Your Product.*

In the next step, you will run your installation.

Running Your Installation

After completing the Project Assistant steps in this tutorial, you have created a fully functional installation that installs the Tutorial executable.

Task ***To run your installation:***

Click the **Run** button on the toolbar or press CTRL+F5.

The installation displays the dialogs that you specified in the Installation Interview page of the Project Assistant. The values you entered in the Project Assistant are displayed to the end user in the appropriate dialogs. For example, at run time, the default value of *INSTALLDIR* that you specified in the Project Assistant appears in the Choose Destination Location dialog box. If the end user browses for a different destination directory, *INSTALLDIR* stores the new value.

After the installation is complete, you can browse for the directory and find the files installed by your installation. If the installation was successful, you will see the tutorial files installed.

Maintenance Mode

When a user runs an installation a second (or later) time for an application installed on their system, the installation runs in *maintenance mode*. Maintenance mode allows the user to modify feature selections from the first-time installation, repair the features already installed, or remove the entire application.

Uninstalling the Application

Task ***To uninstall the Tutorial application:***

Click **Uninstall**.

Now that you have created a basic installation project, click the Installation Designer tab (near the top of the InstallShield window) to expand and fine-tune your installation as illustrated in the next step of the tutorial.

Working in the IDE

After creating a project, you set properties of the project in the InstallShield installation development environment, or *IDE*. The IDE is arranged in functional categories that help you add or edit information in your project. This and later steps of the tutorial explore several of the IDE views.

Note • *The views displayed in the IDE differ, depending on the project type you create.*

After completing this step, you will know how to:

- Set properties for your program features.
- Create components and add file links.

Setting Feature Properties

First, you will set additional properties for the features you created in the Project Assistant, such as the feature display name and description. To edit the feature properties, go to the Features view in the IDE.

Task ***To set feature properties for this tutorial, do the following:***

1. Open the **Features** view. The **Features** view is located in the **Organization** section of the **View List**.
2. In the **Features** view, select the Tutorial_Files feature to display its property grid on the right.

3. Type the following text in the **Description** field: *This feature contains the Tutorial application files.*
4. Select the **New_Feature** feature to display its property grid.
5. Type the following text in the **Description** field: *This feature contains the Tutorial help files*

As you enter the display names and descriptions, InstallShield creates a string entry—displayed as {ID_STRING*n*}—to represent each value.

At run time, if the end user chooses the Custom setup type, the installation displays a dialog that prompts the user to select which features to install. This dialog displays features using the display name and description you specified here.

Creating Components and File Links

You can add links to additional files in the Files and Folders view. In this step, you will add a file to your Help_Files feature. As you add files in the Files and Folders view, the IDE creates components for you according to Setup Best Practices rules.



Task ***To add Tutorial.html to a new component in the Help_Files feature:***

1. Open the **Files and Folders** view.
2. In the **Destination computer's folders pane**, right-click the **Destination Computer** icon and verify that **Show Components** is selected.
3. Select **Help_Files** from the feature list at the top of the view.
4. Expand the tree in the **Destination computer's folders pane** to see the [INSTALLDIR] folder.
5. Right-click the [INSTALLDIR] folder and select **New Component**. Name the component *Help_Component*.
6. In the **Source computer's folders pane**, browse for the Tutorial files source folder containing TutorialHelp.html.
7. Drag the **TutorialHelp.html** icon from the **Source computer's files pane** to the **Help_Component** component in the **Destination computer's folders**. InstallShield adds the file to Help_Component component in the Help_Files feature.
8. Click the **Help_Component** icon to display the component's files in the **Destination computer's files pane**.
9. Because each component should have a key file, right-click the TutorialHelp.html file and select **Set Key File**.

This type of file linking, where the list of files linked to a component does not change, is called *static file linking*. To link to a directory (and possibly its subdirectories) whose contents might change between builds, see Dynamic File Linking.

The next step of the tutorial explains how to build a release image for your installation project.

Building a Release

Before testing an installation, it is necessary to build a release to update your project settings. A release contains all of the files to be distributed to end users on a CD-ROM or from a network location.

The simplest way to build a release is to configure settings in the Release Wizard. The Release Wizard is where you specify release properties, such as the type of media (CD-ROM, for example) to use and whether to compress files on the media.



Task ***To begin using the Release Wizard:***

1. Click the **Release Wizard** button on the toolbar or choose **Release Wizard** from the **Build** menu.
2. In the **Welcome** panel, click Next to begin defining your release settings.

Naming the Product Configuration and Release

Product Configuration Panel

In the Product Configuration panel, you specify the name for the current product configuration. The product configuration name is the name of a folder (inside your project folder) in which your built release will be placed.



Task ***For the tutorial:***

1. Create a new product configuration called *Tutorial*.
2. Click **Next** to specify a release name.

Specify a Release Panel

In the **Specify a Release** panel, specify a release name. The release name is used as the name of a folder (inside the product configuration folder) in which your built release will be placed.



Task ***For the tutorial:***

1. Create a new product release called *CDROM*.
2. Click **Next**.

Specifying Filtering Settings and Languages

Filtering Settings Panel

In the Filtering Settings panel, you can specify features or components to leave out of the current release.



Task ***For the tutorial:***

Use the default settings (no filtering), and click **Next** to continue.

Setup Languages

In the Setup Languages panel, you specify which languages (from among the project languages) the user interface of your installation should display, and whether to display a dialog from which the user can select the installation language.



Task ***For the tutorial:***

Use the default settings (include English in the user interface), and click **Next** to continue.

Selecting the Media Type and Disk Spanning Options

Media Type Panel

In the Media Type panel, you specify the type of media for which you are building a release. The media type you specify indicates the size of the disk image folders the Release Wizard creates. When you build a release for CD-ROM, the Release Wizard divides your disk images into folders, each of which is smaller than 650 MB.



Task ***For the tutorial:***

Select **CD-ROM** from the **Media Type** menu.

Disk Spanning Options Panel

The Disk Spanning Options panel lets you specify how your program files should be arranged, if multiple disk images are required. The Custom spanning type lets you specify the disk image on which to place the files in particular features.



Task ***For the tutorial:***

Select **Automatic**, which lets the **Release Wizard** determine the disk image on which to place each feature's files.

For more information, see Spanning Installations over Multiple Disks or CDs.

Specifying Compression Settings and Setup Launcher Options

Release Configuration Panel

In the Release Configuration panel, you can specify whether to compress all, none, or some of the files in your installation project.



Task ***For the tutorial:***

Select Compress all files.

Setup Launcher Panel

The Setup Launcher panel lets you specify whether to create a Setup.exe setup launcher, and whether to include the Windows Installer 3.1 engine with your installation. The Windows Installer engine installation is necessary for target systems that have older versions of Windows Installer.



Task ***For the tutorial:***

Leave the default settings selected.

For this step, leave all the defaults selected.

Adding Digital Signature and Password Protection

Digital Signature Panel

The Digital Signature panel allows you to digitally sign your application. Digitally signing your application helps to assure your end users that the code within your application has not been modified or corrupted since publication.



Task ***For the tutorial:***

Leave the default settings (no digital signature).

Password & Copyright Panel

The Password & Copyright panel allows you to activate password protection for your installation project and to indicate specific information for your application's copyright.



Task ***For the tutorial:***

Leave the default settings (no password protection or copyright information).

Including .NET Framework Support and Choosing Advanced Settings

.NET Framework Panel

In the .NET Framework panel, specify whether to include .NET Framework support in your release.



Task ***For the tutorial:***

Leave the default settings (do not include .NET Framework).

Advanced Settings Panel

The Advanced Settings panel lets you specify additional settings related to the current release, such as the level of compression to use and whether to create a PDF file for SMS distribution.

Task ***For the tutorial, select the following check boxes:***

- Use long file names
- Optimize size
- Generate Autorun.inf—This generates a file that enables AutoPlay for your CD-ROM image.

For information about the other settings, click the Help button on the Advanced Settings panel.

Reviewing Your Settings

Summary

The Summary panel displays all of the Release Wizard settings for the current release.

Task ***If the settings are correct:***

1. Select the **Build the Release** check box.
2. Click **Finish** to build your release.

Status messages for the build in progress are displayed in the output window. When the build is complete, the files to copy onto the CD are placed in the directory:

`<ProjectFolder>\Tutorial\cdrom\DiskImages\DISK1.`

You can have InstallShield copy your built disk images to another directory using the Events tab in the Releases view.

Rebuilding Your Project

After making changes to your project in later steps of the tutorial, you can rebuild the latest release by clicking the Build toolbar button , choosing Build from the Build menu, or pressing F7.

The next step of the tutorial explains how to create shortcuts and registry data for an installation.

Troubleshooting Your Installation

After running your installation, if files are not installed, check the following parts of your project:

- Verify that **INSTALLDIR** is set to the proper value. This is set in the General Information view.

  For this tutorial, the recommended value is **[ProgramFilesFolder]TutorialCo\TutorialApp**.
- Verify that your features have components and files associated with them.
- After making any changes to your installation, it is necessary to rebuild your project by clicking the Build button or pressing F7.

Step 2: Shortcuts and Registry Data

This step explains how to use the IDE to:

- Create program shortcuts
- Create registry information

The processes for creating other types of system data are similar to the items described in this step. For more information, view the topics listed below.

Creating Shortcuts

You create and modify shortcuts in the Shortcuts view. The properties of a shortcut include its display name, its target executable and arguments, and the icon it displays.

Creating the Shortcut



Task ***In this step you will create a shortcut to Tutorial App in the user's Programs folder, under the Start menu.***

1. Open the **Shortcuts** view. The **Shortcuts** view is located in the **System Configuration** section of the **View List**.
2. Right-click the **Programs Menu** folder icon, and select **New Advertised Shortcut**. The **Browse for a Component** dialog appears.
3. In the dialog, select **Tutorial_Files** from the **Feature** drop-down menu and select Tutorial.exe from the files list and click **Open** to close the dialog.
4. Rename the shortcut icon to an internal name such as *Tutorial*.
5. Set the following shortcut properties:

Table 3-2 • Shortcut Properties

Property	Value	Comment
Display Name	Tutorial Application	To accommodate target systems that do not support long file names, the IDE will create an expression that includes a short file name, as in "TUTORI~1|Tutorial App". If you want, you can modify the short file name part of the expression, as in "TUTORIAL|Tutorial App".
Description	Launch the Tutorial application	Displayed as a tooltip.
Advertised	Yes	At run time, if the user advertises the product or the feature containing the shortcut, the shortcut is created but the component's files are not installed until the user launches the shortcut.

Table 3-2 • Shortcut Properties (cont.)

Property	Value	Comment
Target	Advertised shortcut to [INSTALLDIR]Tutorial.exe	Automatically set to the component's key file for an advertised shortcut.
Icon File	<TutorialSource>\Tutorial.exe	Browse for Tutorial.exe in the source location, and select its only icon.
Icon Index	0	The icon index identifies a particular icon if there is more than one icon resource in the executable file.
Working Directory	[INSTALLDIR]	The working directory should be set to the default directory for your Save As and Open dialogs.



Tip • *To create a shortcut to a file already located on the end user's machine, set the Advertised property to No, and enter the path to the file—using Windows Installer directory properties to represent the path to the file, when possible. For example, to launch a copy of Windows Notepad located in the user's Windows or WinNT folder, enter the shortcut target as [WindowsFolder]Notepad.exe.*

Creating Registry Data

Another common requirement for installations is to write information to the target system's registry. To add registry data to a component, you can use the Registry view.



Task ***To create a registry value called TutorialData under HKEY_LOCAL_MACHINE\SOFTWARE\Tutorial Co\Tutorial\1.00.0000:***

1. Open the **Registry** view.

   The **Registry** view is located in the **System Configuration** section of the **View List**.

2. Select **Tutorial.exe** from the **View Filter** at the top of the view.
3. In the **Destination computer's Registry view pane**, right-click HKEY_LOCAL_MACHINE, select **New**, and point to **Key**.
4. Rename the key *SOFTWARE*.
5. Repeat the process for subkeys named *Tutorial Co*, *Tutorial*, and *1.00.0000*.
6. In the **Destination computer's Registry data pane**, right-click and select **New String Value**.
7. Rename the value *TutorialData*.
8. Double-click the TutorialData value and enter *[INSTALLDIR]* in the Value data field.

The **Registry** view should now appear as follows:

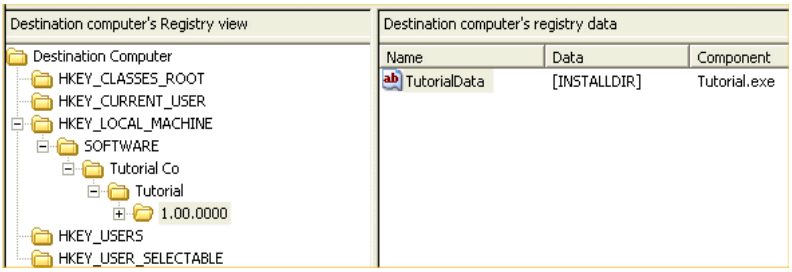


Figure 3-3: Registry View

Tip • *To write the value of a Windows Installer property to the registry, you can use the form [PropertyName]. In this example, creating a registry value whose data is [INSTALLDIR] writes the value of INSTALLDIR to the registry.*

At run time, if the end user selects a setup type or collection of features that includes the Tutorial.exe component, the registry data is created on the target system.

Verifying that the Shortcut Was Created

Task

To verify that your installation created the shortcut:

1. Rebuild your project by clicking the **Build** toolbar button or pressing **F7**.
2. Run the project by clicking the **Run** button or pressing CTRL+F5 (first removing any existing version of the program from your system). A shortcut to the Tutorial application should be present in the **Programs** folder of your **Start** menu.

Verifying that the Registry Data Was Created

Task

To verify that the installation created the registry data:

1. Launch Tutorial App from its shortcut.
2. From the **Tutorial** menu, choose **Verify Registry Data**. If the registry data was created, a message box displaying the value of *INSTALLDIR* is displayed.

The next step of the tutorial explains how to register a COM server (self-registering file).

Step 3: Registering COM Servers

For many files, the installation's only requirement is to copy the files from the source media to the target system. For others, the installer also needs to register the files with the target system. One category of file that needs extra handling is a *COM server*, commonly known as a *self-registering file* or *ActiveX control*. A COM server is usually a DLL or OCX that requires extra information to be written to the target system's registry before applications and Web pages that use the self-registering file can find it.

Creating a COM Server Component

To install and register these and other types of files, you can use the Component Wizard. The Component Wizard creates components that copy the files, and perform any additional registration steps. In this step you will create a component that installs and registers Tutorial.ocx and an HTML file that uses it.

Task ***To create a COM server component:***

1. Open the **Files and Folders** view. The **Files and Folders** view is located in the **Application Data** section of the **View List**.
2. At the top of the **Files and Folders** view, in the **View Filter** list, select the **Tutorial_Files** feature.
3. In the **Destination computer's folders pane**, right-click the **[INSTALLDIR]** folder and select **Launch Component Wizard**.
4. In the **Welcome** panel of the **Component Wizard**, select the Let me select a type and define the component myself option and click **Next**.
5. In the **Component Type** panel, select the **COM Server** icon, type *Tutorial.ocx* in the **Component Name** field, and click **Next**.
6. In the **COM Server—Destination** panel, verify that the destination is set to [INSTALLDIR].
7. In the **COM Server File** panel, click the browse button next to the **COM Server File** field and browse for Tutorial.ocx in your tutorial files source directory. Click **Next**.
8. After the Component Wizard has extracted the COM information, review the COM information and click **Finish** to create the component.

The next step is adding the HTML file to the component you just created.

Task ***To add the HTML file to the component:***

1. In the **Destination computer's folders pane** of the **Files and Folders** view, select the new Tutorial.ocx component.
2. Drag the file TutorialCtrl.html from the **Source computer's files pane** to the **Destination computer's files pane**.
3. Verify that Tutorial.ocx is marked as the key file of its component.

Note • *See Registering COM Servers for other options for registering self-registering files, including extracting COM information each time that you rebuild the release—for COM servers with interfaces that change between builds—or calling the file's self-registration functions.*

Verifying that the COM Server Was Registered

Task ***After rebuilding your release (by pressing F7) and running the installation (by pressing CTRL+F5), you can verify that the COM server was registered properly:***

1. Launch Tutorial App using its shortcut in the **Programs** menu, or by double-clicking its icon.
2. Choose **COM Server Test** from the **Tutorial** menu.
3. If the COM server was registered correctly, the HTML page displays a "success" message.

The Component Wizard can also create components that install and configure fonts and Windows NT services.

The next step of the tutorial demonstrates how to install files conditionally.

Step 4: Conditions and Properties

In this step, you will learn how to conditionally install data on a target system.

Operating System Conditions

A common requirement for installations is to install certain files on a system only if particular conditions are met. For example, files may be specific to an operating system or language, or should be installed only if the user has appropriate privileges.

To install a component (and its files and other data) only on particular operating systems, you can use the component's Operating Systems property. You can modify a component's properties by opening the Setup Design view, expanding the feature icon that contains the feature, and selecting the desired component.

Task ***To create a component that will be installed only on systems running Windows 7 or later:***

1. Open the **Setup Design** view. The **Setup Design** view is located in the **Organization** section of the **View List**.
2. Right-click the **Help_Files** feature and select **New Component**.
3. Rename the component *Windows_7_Files*.
4. Expand the Windows_7_Files component, click the **Files** icon for the component, and add the file ReadmeNT.txt from your tutorial files source folder by right-clicking in the Files pane and browsing to the file.
5. Right-click the .txt file and select **Set Key File**.
6. Click the Windows_7_Files component to display the component's property grid.
7. Select the component's Condition property and click the browse button to launch the **Condition Builder** dialog.
8. Create the following condition: *VersionNT>=601*. For information on creating conditions, see Building Conditional Statements.
9. Click **OK** to close the **Condition Builder** dialog and add the condition.

After you rebuild (by pressing F7) and run the installation (by pressing CTRL+F5), and any files or other data contained in the component will be installed only if the target system is running Windows 7 or later.

Windows Installer Conditions

The Windows Installer service stores some global information about your installation and about the user's operating system in properties. Some properties are built into the Property table of your MSI database, and some are created and set by the Windows Installer engine when the user launches an installation.

Properties commonly used in conditions include:

- AdminUser, which is set if the user running your installation has administrative privileges.
- VersionNT, numeric values describing the operating system version the user is running.
- PhysicalMemory, which contains the amount of RAM—in megabytes—on the user's system.

A Windows Installer condition is a statement of logic that compares a property value against a constant value, or tests if a property exists. For example, Windows Installer defines properties called ScreenX and ScreenY, which contain the user's monitor resolution in pixels. A Windows Installer condition that checks that the user has at least 800 by 600 resolution would read "(ScreenX>=800) And (ScreenY>=600)".

Conditions can also test if a property is defined. For example, the AdminUser property is set only if the user has administrative privileges, and a condition that tests if a user has administrative privileges is simply "AdminUser".



Task ***To create a component that will be installed only if the user has administrative privileges:***

1. Right-click the **Help_Files** feature and select **New Component**.
2. Rename the component *Admin_Component*.
3. Expand the **Admin_Component** component and click the **Files** icon.
4. Add the file AdminOnly.txt from your tutorial files source folder, and set it as the key file of the component.
5. Click the browse button in the component's **Condition** property to display the **Condition Builder** dialog.
6. In the **Condition Builder** dialog, type *AdminUser* in the **Condition(s)** field.
7. Click **OK**.

At run time, the component's data are installed only if the user has administrative privileges.

The next step of the tutorial describes how to modify your installation's user interface.

Step 5: Changing the End-User Interface

This step describes three ways you can modify the end-user interface of your installation:

- Specifying the dialogs to be displayed during the installation.
- Modifying the layout and properties of a dialog using the Dialog Editor.

Adding a New Dialog

The Basic MSI project includes many dialogs that you can display in your installation's user interface. The topic, Running Your Installation, in this tutorial, showed the dialogs your installation displays based on your selections in the Project Assistant's Installation Architecture page.

Task ***To create a new dialog:***

1. Open the **Dialogs** view. The **Dialogs** view is located in the **User Interface** section of the **View List**.
2. Right-click the **All Dialogs** explorer and then click **New Dialog**. The **Dialog Wizard** opens. Click **Next** to dismiss the **Welcome** panel.
3. In the **Dialog Template** panel, click **Interior Wizard Panel**, and select the **Let me insert this dialog in a User Interface sequence** check box.
4. In the **User Interface** panel, select **Installation** in the **User Interface Sequence** list. In the list of dialogs, select **InstallWelcome**. Based on these selections, InstallShield will insert your new dialog in sequence immediately following the **InstallWelcome** dialog.
5. In the **Dialog Position and Condition** panel, leave the default settings, and click **Finish**. Your new dialog appears in the **Dialogs** list.
6. Right-click the dialog and select **Rename**. Rename the dialog **WelcomeBitmap**.

Using the same technique, you can insert additional dialogs in your installation's user interface.

Modifying Dialog Layout in the Dialog Editor

The Dialog Editor allows you to modify the appearance of dialogs displayed by your installation.

Task ***In this step, you will modify the WelcomeBitmap dialog that you just created:***

1. First, create a bitmap (using a program like Microsoft Paint) that measures 300 by 150.
2. Open the **Dialogs** view.
3. Expand the WelcomeBitmap dialog's node. Click **English (United States)** to open the **Dialog Editor**.
4. Click the **Dialog Bold Title** text box at the top of the dialog. In the **Text** field, type *Welcome Bitmap*. This changes the dialog's main title.
5. Click the **Dialog Normal Description** text box at the top of the dialog. In the **Text** field, type *Displays my welcome bitmap*. This changes the dialog's description.
6. Click the **Bitmap** button on the **Dialog Control** toolbar and use the cursor to drag a box on the dialog. Set the Height to 150 and the Width to 300.
7. In the **File** field browse to the bitmap file that you created in step 1.

After rebuilding the project (by pressing F7) and running it (by pressing CTRL+F5), the Welcome Bitmap dialog will appear after the Install Welcome dialog.

Tip • *To maximize the Dialog Editor view, press CTRL+M.*

Globalization Tutorial

The Globalization tutorial introduces you to the tools and options that InstallShield provides for creating a global installation package. A global installation is one that has the potential to run in many different languages. Depending on how you choose to build your installation, you can either include all the languages in one package and let the end user select the language, or you can build individual installation packages for each language that you target. This tutorial walks you through the process of creating an all-encompassing installation.



***Edition** • The Premier edition of InstallShield must be installed on your development system in order to successfully complete this tutorial.*

If you have not already done so, you may want to run through the Basic MSI tutorial to familiarize yourself with how to create an installation package. When finished with the Basic MSI Tutorial, you will have a Basic MSI installation project that is ideal for adding additional languages. The complete project file for the Basic MSI tutorial was installed with this product in one of the Samples subfolders within the InstallShield Program Files folder. The default location is:

C:\Program Files\InstallShield\2016\Samples\WindowsInstaller\Tutorial Project\Tutorial



***Project** • Almost all of the information in the tutorial also applies to InstallScript installation projects. Differences are explicitly noted in the tutorial text.*

Opening the Project File

This tutorial uses **Othello.ism**, which is a sample project file that is installed with InstallShield.



Task ***To begin the tutorial:***

Open **Othello.ism**, which is located in one of the Samples subfolders within the InstallShield Program Files folder. The default location is:

C:\Program Files\InstallShield\2016\Samples\WindowsInstaller\Basic Installation Project



***Project** • Othello.ism is a Basic MSI installation project. Almost all of the information in this tutorial also applies to InstallScript installation projects; differences are explicitly noted in the tutorial text.*

Selecting the Target Languages

The first step in creating a global installation is to select the languages that you want to target. For this tutorial, you will add two languages to your installation project: German and Polish.

Task ***To add languages to your project:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Setup Languages** setting, click the ellipsis button (**...**). The **Setup Languages** dialog box opens.
3. Select the **English (United States)**, **German**, and **Polish** check boxes.

InstallShield adds string entries for English, German, and Polish to your project. Each string entry consists of a language-independent identifier and a corresponding language-specific value. The string entries include the built-in user-interface string resources that are already translated. To view all of the string entries, you can use the String Editor view. (In the View List under User Interface, click String Editor.)

Every time that you add a new string entry to your default language, a parallel entry is made to the string entries for all of the other languages that are in your project.

Editing Language-Specific String Entries

The next step is to edit string entries in your installation project. You will edit string entries for three different settings: The feature's display name, the shortcut's display name, and the shortcut's description. (The shortcut's description will be created as part of the next step in this tutorial.)

Feature Display Name

Task ***To provide a display name for a feature:***

1. In the View List under **Organization**, click **Setup Design**.
2. In the **Setup Design** explorer, click **ProgramFiles**.
3. In the **Display Name** setting in the right pane, enter **Program Files** if it is not already there.

If you click anywhere else within the Setup Design view, you will see that the text that you entered has been preceded by a string identifier, which is enclosed within curly brackets ({}). You can view all of your project's string entries by clicking the ellipsis button (...) that is displayed when you click the Display Name setting.

Shortcut Display Name

Task ***To add a display name for the shortcut:***

1. In the **Setup Design** explorer, expand the **ProgramFiles** feature, and then expand the **Program_Executables** component.

   The component Program_Executables contains a shortcut to the file **Othello.exe**.
2. Click the **Shortcuts** icon under the **Program_Executables** component.
3. In the **Shortcuts** explorer, click the **Othello** shortcut. InstallShield displays the shortcut's settings in the right pane.

Currently, the **Display Name** setting is set to **Othello**. The value is preceded by a string identifier, which is enclosed within curly brackets ({}).

4. To change the display name, enter the new name in the **Display Name** setting.

Creating String Entries

Because many of your text strings might be used in multiple places in your project, it is inefficient to store each instance of these strings in the project. Instead, you can create the string once, and use it anywhere a string is needed. To help streamline the process of localizing a project, all of the text strings that may be displayed at run time during the installation process are available in one consolidated view: the String Editor view. You can use this view to create new string entries.

For the shortcut's Description setting, you are going to enter your new string entry through the String Editor view. Then, you will associate that string with the shortcut's description.



Task ***To create a new string entry and use it for the shortcut's Description setting:***

1. In the View List under **User Interface**, click **String Editor**.
2. Do one of the following:
   - Click the **New** button.
   - Press the INSERT key.

   The **String Entry** dialog box opens.
3. In the **String Identifier** box, enter the following:

   MYSTRING
4. In the **Value** box, enter the following text:

   This is the description of the Othello shortcut.
5. In the **Comments** box, you can optionally specify an internal note about the string entry. The comments are not displayed at run time.
6. Click **OK**. InstallShield adds new rows in the String Editor view, one for each language (English, German, and Polish).
7. In the View List under **System Configuration**, click **Shortcuts**.
8. In the **Shortcuts** explorer, click the **Othello** shortcut. InstallShield displays the shortcut's settings in the right pane.
9. In the **Description** setting, click the ellipsis setting (**...**). The **Select String** dialog box opens.
10. Select the **MYSTRING** row, and then click **OK**.

Your new string is now entered as the value for the **Description** setting.

Including Language-Specific Files and Components

The next step in creating a global installation involves including language-specific files and components within your installation project. For example, although many of your program files may be language-independent, your help files and run-time strings are both language-specific. For the purpose of this exercise, you will add three new components to your project. Each component contains a Readme file that is localized in all of your supported languages.



Task ***To add a component to your project:***

1. In the View List under **Organization**, click **Setup Design**.
2. In the **Setup Design** explorer, right-click the feature called **ProgramFiles**, and then click **New Component**.
3. Type *English_Readme* for the component's name.
4. Repeat this process two more times. Name these components *Polish_Readme* and *German_Readme*.

Translated versions of a simple Readme file are included with InstallShield. These files are located in one of the Samples subfolders within the InstallShield Program Files folder:

*`InstallShield Program Files Folder`***\Samples\WindowsInstaller\Basic Installation Project\Data Files\Readme**



Task ***To add a file to the English_Readme component:***

1. Expand the new **English_Readme** component, and then click the **Files** icon under the **English_Readme** component.
2. Right-click in the **Files** list and click **Add**.
3. Browse to the **English.txt** file, and then click **Open**.

Repeat these steps for the German and Polish components, adding **Deutsch.txt** to and **Polski.txt**, respectively.

Specifying Component Installation Conditions

Now that you have created your language-specific components, you need to include logic that will let the installer know which of these components should be installed. By specifying a component condition, you can determine the default language of the target system and then install the appropriate file. Each of the three language-specific components that you created will need a condition. If that condition evaluates to True, the component is installed.



Task ***To create the component condition:***

1. Click the German_Readme component.
2. In the right pane, click the **Condition** setting, and then click the ellipsis button (**...**) for this setting. The **Condition Builder** dialog box opens.
3. In the **Properties** list, select **SystemLanguageID**, and then click the **Add** button.
4. In the **Operators** list, select the equals sign (**=**), and then click the **Add** button.

The **Condition(s)** box contains **SystemLanguageID =**, which reflects the selections you previously made.

5. Next you need to provide a value that will be checked when the installation is run. Because you are currently editing the German component, enter **1031** after the equal sign. 1031 is the language ID for German. Since the component is installed only if this equation evaluates to true (that is, the target system's language is German), this component is not installed on any machine that is not running in German.

Follow the same steps from above to add a condition to the Polish_Readme component. Instead of using 1031 as the language value, use 1045, which is the language ID for Polish.

You need to choose one language as your default language. For this example, English is the default. Therefore, the condition that you use for the English_Readme component differs from the other two. The condition for the English_Readme component should appear as follows:

```
SystemLanguageID<>1045 AND SystemLanguageID<>1031
```

With this logic, if the language of the target machine is not German or Polish, the English_Readme component is installed.



Project • *To learn how to specify which language-dependent components are installed at run time for InstallScript and InstallScript object projects, see Installing Components Based on Language.*

Translating the Strings

Before you can build your installation project, you need to translate the English strings that you entered for the feature's display name, the shortcut's display name, and the shortcut's description. For this tutorial, they have been translated for you. All you have to do is enter the correct text for the German and Polish string entries. You can use the String Editor view to do this.

German String Values

Open the String Editor view. Find each of the following string identifiers for the German language. You can enter the string identifier in the Search Grid box to easily find each string. As an alternative, you can click the Identifier column heading to sort all of the string entries by identifier. Update the German values for the three string entries as follows:

Table 3-1 • German String Entries

Identifier	Value
FEAT_DISPLAYNAME	Programmdateien
SHORTCUT_DISPLAYNAME	Othello-Verknuepfung
MYSTRING	Dies ist die Beschreibung der Verknuepfung fuer Othello.

Polish String Values

Update the Polish values for the three string entries:

Table 3-2 • Polish String Entries

Identifier	Value
FEAT_DISPLAYNAME	Pliki Programu
SHORTCUT_DISPLAYNAME	Skrt do Othello
MYSTRING	Opis skrtu Othello.

In most translation situations, you export your string entries for translation. To learn more, see Translating String Entries. However, for the purposes of this tutorial, it is easier to edit the string entries within the String Editor view.

Building the Installation

Your installation project has been fully globalized and is ready to test. Before you can test the installation, however, you need to build it.



Task ***To build your installation:***

1. Click the **Release Wizard** button on the toolbar.
2. In the first two panels, specify a product configuration and release name.
3. Accept the default settings for every other panel in the wizard except the **Setup Languages** panel.
4. The **Setup Languages** panel enables you to choose which languages to include in your installation. Only the languages that you specified in the General Information view appear in the list of available languages—English, Polish, and German. Select the check box next to each language.
5. Also, ensure that you select the **Display the Setup Languages Dialog** check box. This dialog allows end users to select the language in which they want the installation to run.
6. Accept the default settings provided in the remaining wizard panels.
7. When you reach the **Summary** panel, ensure that the **Build the Release** check box is selected, and then click the **Finish** button.
8. InstallShield builds the release.

Running the Installation

Task ***To run your installation:***

1. Click the **Run Setup** button on the toolbar. The **Choose Setup Language** dialog appears.
2. To run your installation, select **German (Standard)** and click **OK**. From this point forward, every dialog is displayed in German.

Note *• After an installation is run in a particular language, Windows Installer caches this information and always runs the installation in that language.*

As the installation wizard progresses, you may notice that some buttons are not properly sized. You can easily fix this problem by opening the Dialog Editor and resizing the controls so the text fits in the control.

In the Custom Setup panel (in German it is called Angepasstes Setup), the feature name is now *Programmdateien*. Your localized string entries are a part of the installation.

Testing the Installation

The last step is testing the globalized installation that you created.

Task ***To test the installation:***

1. Open the **Start** menu and select **Programs**. The Othello shortcut is displayed as Othello Verknpfung. You can see the shortcut's description, which also appears in German.
2. Navigate to the installation directory for Othello. It should be in <Program Files Folder>\Shakespeare Inc\Othello. The readme file that you installed is called Deutsch.txt.

4

Creating Installations

If you have ever installed an application onto your computer, you have seen an installation in action—from the end user's perspective. An installation's primary task is to transfer files from the source medium to the local drive. An installation often also displays a user interface to obtain end user selections, configures the target system (for example, makes any required registry entries and creates shortcuts), and enables modification or uninstallation of the installed application. Creating an installation involves performing some or all of the following tasks.

Specify Installation Information

Basic information that you enter in the General Information view is used in various parts of the installation; for example, the product name is used to create the application information registry key.

Organize and Transfer Files

File transfer involves copying files from the source medium, such as a CD or DVD, to a local drive on the end user's machine. Depending on the configuration the end user chooses—by selecting a setup type (in an InstallScript or InstallScript MSI installation) or features—all or only some of the files may be transferred to the local disk.

Organize the files to be installed into setup types and features to help your end users select the most appropriate files. Within each feature, organize the files into components according to their type and purpose, for example, files that are installed to the same target folder.

Configure the Target System

In addition to installing files, many installations need to configure the target system by creating shortcuts and program folders, modifying the registry, modifying initialization file (.ini file) data, configuring Open Database Connectivity (ODBC) resources, modifying environment variables, modifying XML files, modifying text files, schedule tasks, and install and control Windows services.

Customize Installation Behavior

InstallShield offers wide-ranging customization options. InstallScript installations are driven by InstallShield's simple but powerful InstallScript programming language, which—in addition to its built-in functions—enables you to call DLL and Windows API functions and launch child installations and other applications from your installation. Windows Installer-based installations can use custom actions to run InstallScript, VBScript, or JavaScript code; call

DLL functions; run executable files; call a managed method in a managed assembly; set a property or a directory; trigger an error and end the installation; run PowerShell scripts; terminate a process; or run other installation packages.

Define the End-User Interface

An installation's end-user interface provides information and installation configuration options to the end user. Through the user interface, an end user can choose to install only part of a product, choose to leave some files on the source medium, view a license agreement, or provide the installer information that may be necessary to properly configure the installation.

The user interface can be customized to meet the needs of your installation. For example, you can prompt a user for a serial number before starting the installation to protect your software against illegal use. During file transfer, an installation can display billboards that provide product information such as new features or usability tips. A status bar may also be displayed to show the progress of the file transfer process.

Configure Servers

A server-side installation may need to create and manage new Internet Information Services (IIS) Web sites, manage COM+ applications and components, or manage and organize SQL scripts by server connections and settings.

Prepare Installations for Update Notification via FlexNet Connect

FlexNet Connect lets you automatically notify your Web-connected end users when patches, updates, and product information for your application are ready for release. To take advantage of FlexNet Connect, you must enable FlexNet Connect in your original installation.

Prepare the Installation for Maintenance and Uninstallation

To uninstall, modify, or repair an application, the operating system must have some indication that the application is present. To accommodate this, an installation registers an application with the operating system so that it can be easily maintained or uninstalled.

Much of the information registered in this process is available to the end user through Add or Remove Programs in the Control Panel. For example, technical support contact information, product update information, product version, and product publisher information are registered in this process.

Build, Test, and Deploy the Installation

Once you have created your installation project, you will want to build, test, and deploy the installation: create the files that you will release to your users, test the installation for errors, and optionally copy the files to a local or network location or an FTP site.

Before You Begin

The "Before You Begin" section of the InstallShield Help Library contains information that is helpful to installation authors as they create new installation projects with InstallShield. The topics provide background information on Windows Installer, the Windows logo program, INSTALLDIR, TARGETDIR, and other areas of installation development.

Requirements for the Windows Logo Program

Microsoft established a list of requirements that a product and its installation must fulfill in order to participate in the Windows 8 Desktop App Certification Program. The requirements outline criteria that help make a product more compatible, reliable, and secure when running on Windows systems. Products that meet the Windows 8 Desktop App Certification Program requirements can carry the Compatible with Windows 8 logo.

Qualifying for the Windows Logo Program

To learn how to qualify for the Windows logo program, visit MSDN. This Web site has information about the Windows logo program.

Certified for Windows Validation Suites in InstallShield

Validating your installation package or merge module may help you identify whether your product meets installation requirements for Windows logo program. If a package or merge module fails one or more validation rules, InstallShield reports the specific rules that were violated and offers additional information to help you troubleshoot the problem.

Therefore, if you are interested in being able to use the Windows logo, consider using both of the following suites to validate your installation package:

- **InstallShield Validation Suite for Windows 8**—This suite consists of a number of InstallShield internal consistency evaluators (ISICEs) that help you identify issues that may make your installation behave unexpectedly on Windows systems. This suite checks for issues that may not be revealed in the Full MSI Validation Suite.
- **Full MSI Validation Suite**—This suite consists of ICEs that Microsoft created.

If you create a merge module in InstallShield, use the following suites to validate your merge module:

- **InstallShield Merge Module Validation Suite for Windows 8**—This suite consists of a number of InstallShield ISICEs that help you identify issues that may make your merge module behave unexpectedly on Windows systems. This suite checks for issues that may not be revealed in the Merge Module Validation Suite.
- **Merge Module Validation Suite**—This suite consists of ICEs that Microsoft created.

Additional validation suites are also available.

To learn more, see Validating Projects.



Windows Logo • *The Windows Logo Guideline alert appears throughout the InstallShield Help Library whenever the information relates to complying with the Windows logo program guidelines.*

Introduction to Windows Installer

A Windows Installer–based installation is distributed as an .msi package, which consists of a Windows Installer database (.msi database) and related data files (.cab files, uncompressed data files, etc.). The .msi databases, implemented as COM structured storage, contain dozens of tables that describe the changes that are to be performed on the target system. For example, some of the .msi tables are:

- **File**, which describes the files to be installed

- **Registry**, which describes the registry data to be written
- **Shortcut**, which describes shortcut settings

Other .msi database tables describe the appearance and behavior of the installation's user interface, install and configure Windows services and ODBC information, determine characteristics of the target system, and store icons and other binary data for use during installation.

From a developer's perspective, perhaps the greatest change in Windows Installer installation programs is that there is no explicit script to write. Instead, Windows Installer–based installations perform standard and custom actions, where an action displays a dialog, queries the target system, or makes changes to the target system. These actions are arranged into sequences, which are ordered collections of actions.

Windows Installer includes a collection of application program interface functions, or APIs, dedicated to managing product installations. Applications must call the Windows Installer APIs in order to take advantage of features available with Windows Installer.

An integral part of Windows operating systems, Windows Installer provides a standard format for component management as well as an interface for managing applications and system tools. Various versions of Windows Installer are available as redistributables for Windows operating systems.

Although it is possible to create a Windows Installer package by editing .msi database tables directly, the large number of tables and relationships among them makes doing so a formidable task. InstallShield organizes the process of developing an installation for Windows Installer into various views, providing graphical editors and wizards that shield the developer from much of the implementation detail that is associated with .msi databases.

For more information on Windows Installer technology, see the Windows Installer Help Library.

Minimizing the Number of User Account Control Prompts During Installation



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Some of the specific details apply to only some of these project types. These differences are noted where appropriate.

One of the goals of Windows Vista and later, as well as User Account Control (UAC), is to allow users to run as standard users all of the time. Elevation should rarely be required; if it does occur, it should occur for as short of a duration as possible.

Several different areas of InstallShield affect whether an installation triggers UAC consent or credential prompts for elevated privileges. Understanding these different settings will help you create the appropriate UAC experience for your installation when end users run it on Windows Vista and later systems. It will also offer guidance if you are trying to minimize the number of UAC prompts that are displayed during your installation.

Depending on how it is configured, an installation that includes InstallShield prerequisites may prompt for elevated privileges on Windows Vista and later systems at several different points during the installation:

1. When the end user launches the **Setup.exe** file
2. When the **Setup.exe** file launches a setup prerequisite that requires elevated privileges
3. When the ISInstallPrerequisites custom action relaunches the **Setup.exe** file in feature prerequisite installation mode because one or more of the features being installed has an associated feature prerequisite

   Note that the ISInstallPrerequisites custom action does not verify whether any feature prerequisites require elevated privileges before the prompt for elevated privileges is displayed. In addition, the ISInstallPrerequisites custom action does not check the conditions on any of the feature prerequisites to determine whether the feature prerequisites need to be installed. The prompt for elevated privileges is always displayed.

   Project • *Basic MSI projects include support for feature prerequisites.*
4. When the Windows Installer begins the Execute sequence of the .msi package

   Project • *This last installation point applies to Basic MSI and InstallScript MSI projects, but not InstallScript projects.*

UAC-Related Settings in InstallShield

Following is a list of the InstallShield settings that help determine whether UAC prompts are displayed during an installation on Windows Vista and later systems:

- **Required Execution Level**—Use this setting in the Releases view to specify the minimum execution level required by your installation's **Setup.exe** file. InstallShield uses the value that you select (Administrator, Highest available, or Invoker) in the application manifest that it embeds in the **Setup.exe** launcher. For more information, see Specifying the Required Execution Level for Your Setup Launcher on Windows Vista and Later Platforms.
- **The prerequisite requires administrative privileges**—If you are creating or modifying an InstallShield prerequisite for your installation, use this check box to indicate whether administrative privileges are required in order to install this prerequisite. This check box is on the Behavior tab in the InstallShield Prerequisite Editor. For more information, see Specifying that an InstallShield Prerequisite Requires Administrative Privileges.
- **Require Administrative Privileges**—Use this setting in the General Information view to specify whether the Execute sequence of your installation's .msi package requires administrative privileges. If you set this to No, InstallShield sets bit 3 in the Word Count Summary property to indicate that elevated privileges are not required to install your product. For more information, see Entering Summary Information Stream Data.

  Project • *The Require Administrative Privileges setting does not apply to InstallScript projects.*
- **Advertise If Prerequisites Are Elevated**—Use this setting in the Releases view to specify whether the .msi package should be advertised—and if so, whether it should be run silently or with the full user interface (UI)—after the InstallShield prerequisites in the installation have been successfully installed with elevated privileges on Windows Vista and later machines. The advertisement may allow end users to avoid the UAC prompt that would otherwise be displayed for an .msi package that requires elevated privileges. For more information, see

Specifying Whether a Product Should Be Advertised If Its InstallShield Prerequisites Are Run with Elevated Privileges.



Project • *The Advertise If Prerequisites Are Elevated setting does not apply to InstallScript projects.*

In addition, the type of InstallShield prerequisite—either setup prerequisite or feature prerequisite—may affect whether UAC prompts are displayed during an installation on Windows Vista and later systems. To learn more about these two types of InstallShield prerequisites, see Setup Prerequisites vs. Feature Prerequisites.

Note the following UAC-related behavior on Windows Vista and later:

- If Required Execution Level is set to Invoker, any InstallShield prerequisites in your installation do not require administrative privileges, and Require Administrative Privileges is set to No, end users should see no UAC prompts during installation.
- If Required Execution Level is set to Invoker, your installation includes setup prerequisites that require administrative privileges, and Require Administrative Privileges is set to No, end users should see one UAC prompt—plus up to one additional UAC prompt for each reboot—during installation.
- If the full user interface of the setup launcher is displayed and the installation includes setup prerequisites that need to be installed, the setup launcher typically displays the setup prerequisite dialog before the main installation starts. If one or more of the setup prerequisites that need to be installed require administrative privileges, the Install button on the message box has the shield icon to alert the end user that elevated privileges are required.
- If the installation is continuing after a reboot and privileges must be elevated, the OK button of the continuation message box has the shield icon. If privileges do not need to be elevated, the shield button is not displayed.
- If your installation includes more than one setup prerequisite that must be installed on a target machine and one or more of those setup prerequisites requires administrative privileges, the UAC prompt is displayed before the first setup prerequisite is installed. This may allow elevated privileges to be used for all prerequisites without requiring separate UAC prompts for each prerequisite installation. Note, however, that if a setup prerequisite installation causes a reboot, administrative privileges are lost, and a UAC prompt may be displayed if any of the remaining prerequisites require administrative privileges.

  A slightly different behavior applies to feature prerequisites. If your installation is going to install any features that are associated with prerequisites, the UAC prompt is displayed when the ISInstallPrerequisites custom action relaunches **Setup.exe** in feature prerequisite installation mode. This occurs regardless of whether any of the feature prerequisites require elevated privileges. It also occurs before any of the feature prerequisites' conditions are evaluated to determine whether the feature prerequisites need to be installed. Note that if a feature prerequisite installation causes a reboot, administrative privileges are lost. After the reboot, the ReadyToInstall dialog is displayed again, and the end user needs to click the Install button to proceed with the rest of the installation. In this case, the UAC prompt is displayed again when the ISInstallPrerequisites custom action relaunches **Setup.exe** in feature prerequisite installation mode.
- Note that if Require Administrative Privileges is set to No but your .msi package tries to perform a task for which it does not have adequate privileges, Windows Installer may display a run-time error.
- If privileges are elevated at the end of an installation and the SetupCompleteSuccess dialog launches the product, elevated privileges are carried over to your product. In most cases, running an application with elevated privileges is discouraged.

Sample Scenarios

The following sections contain examples that illustrate different combinations of values for the aforementioned settings in InstallShield. The diagrams show when Windows Vista and later request elevated privileges for standard users or administrative users who have limited privileges. The examples are based on the default UAC settings on Windows Vista and later systems.

Example 1: UAC Prompt Is Displayed for a Prerequisite that Requires Administrative Privileges; .msi File Is Advertised

The example 1 diagram shows an installation that requires elevation for a setup prerequisite, a feature prerequisite, and for the Execute sequence of the .msi package. Windows Vista and later display one UAC prompt for the setup prerequisite, and another one for the feature prerequisite.

If the feature prerequisite was not included or if it was a setup prerequisite instead of a feature prerequisite, the second UAC prompt (which is labeled as *UAC prompt #2* in the diagram) would not be displayed. In these cases, UAC prompt #2 would not be needed because the .msi package is successfully advertised after the setup prerequisite is installed with elevated privileges.

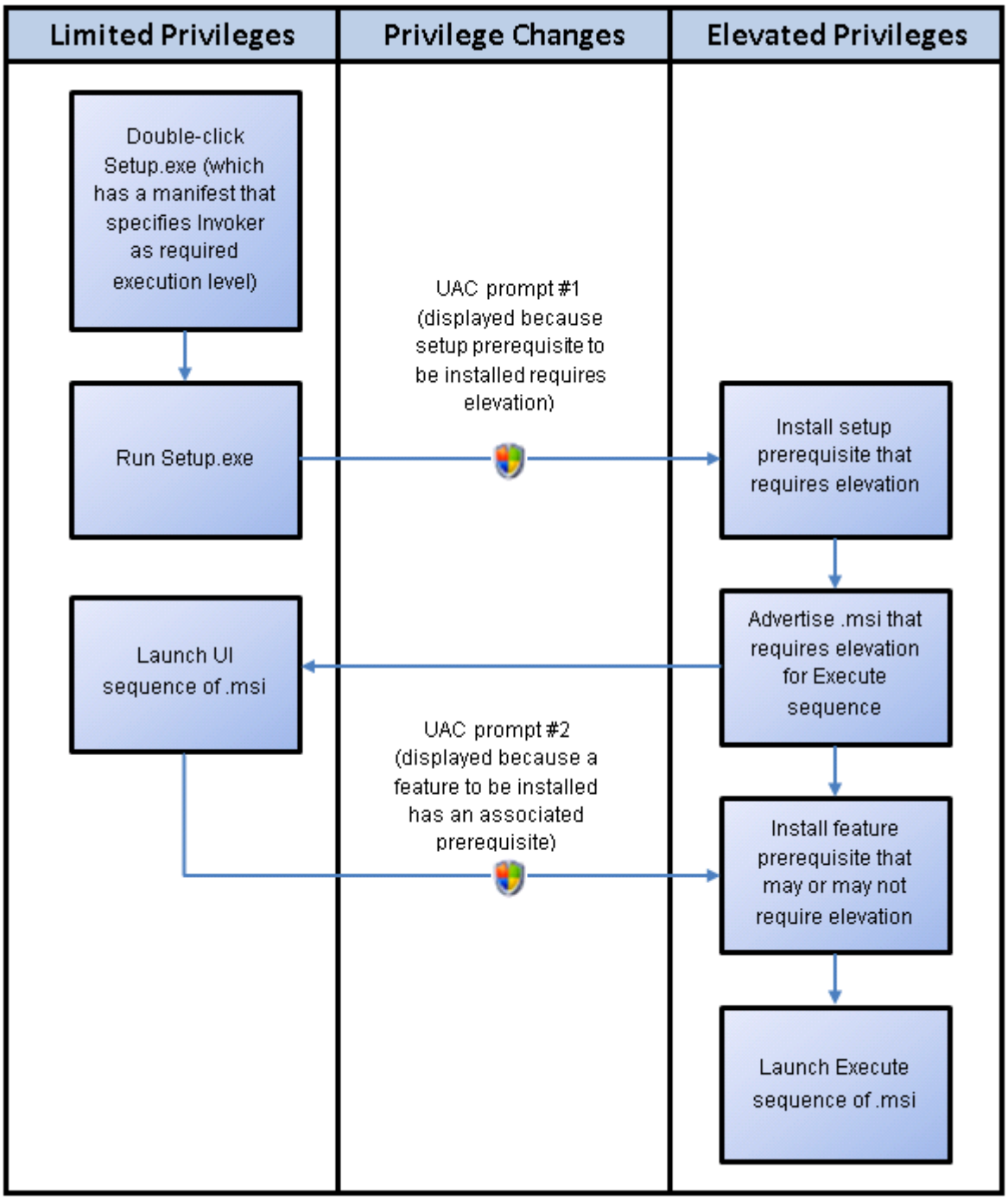


Figure 4-1: Example 1—Diagram of an Installation that Has Invoker as the Required Execution Level and that Advertises the .msi Package

Example 2: UAC Prompt Is Displayed for Setup.exe and After a Reboot for a Prerequisite that Requires Administrative Privileges

The example 2 diagram shows an installation that requires elevation for **Setup.exe**, two setup prerequisites, a feature prerequisite, and the Execute sequence of the .msi package. Because the **Setup.exe** file has a manifest that specifies Administrator as the required execution level, elevated privileges are used for each part of the installation. The second UAC prompt is displayed because elevated privileges are lost during a reboot.

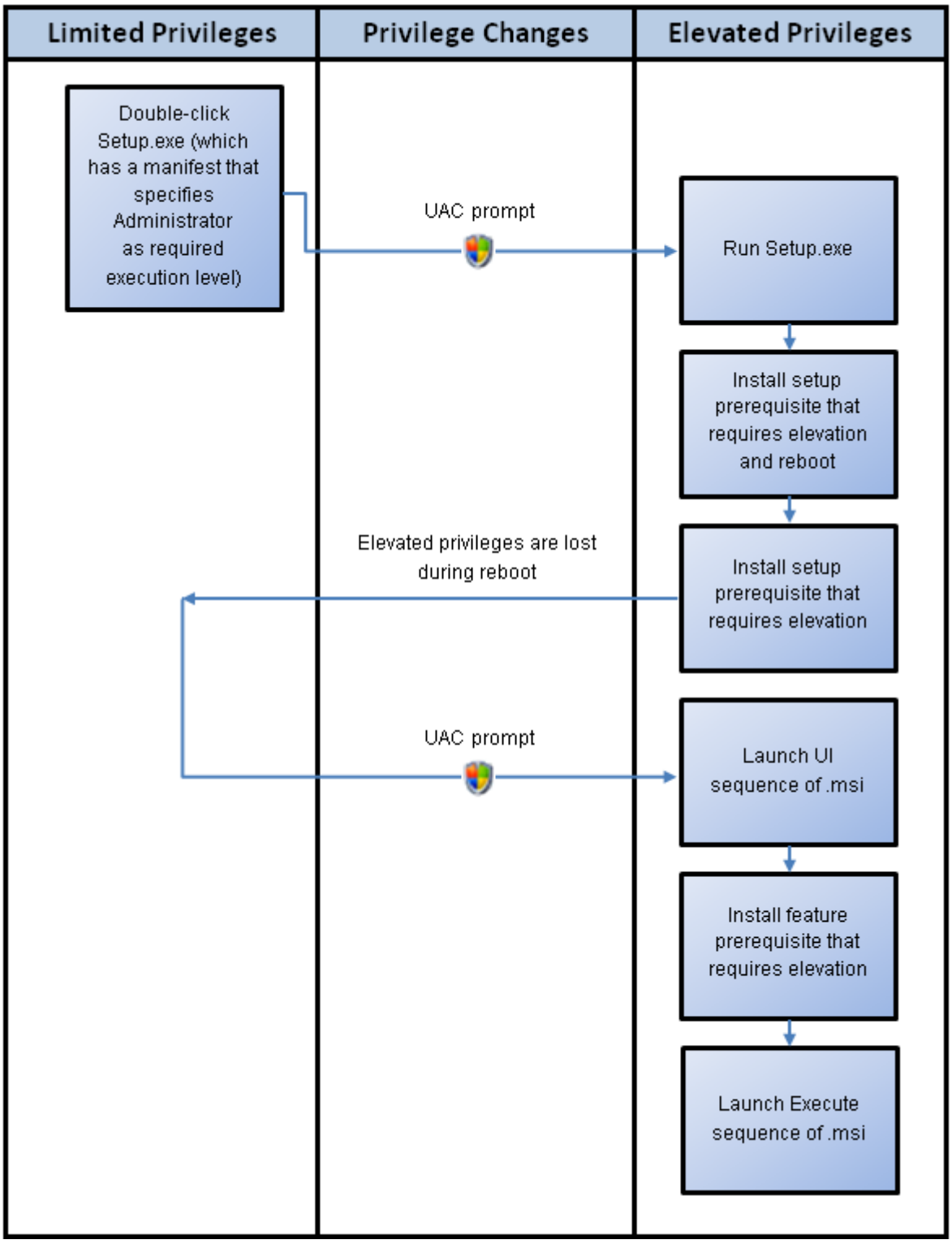


Figure 4-2: Example 2—Diagram of an Installation that Has Administrator as the Required Execution Level

Setting an Application's Disk Usage

Windows Installer uses a process called file costing to determine the total disk space a current installation requires. This encompasses costs for files that will be installed or removed, registry entries, shortcuts, and other components of an installation.

File Costing

File costing takes into account the fact that some files are overwritten by newer versions, which decreases the file cost. These values are dependent on the volume to which each file is to be installed or removed, and they are recalculated when a component's directory association is changed.

File costs are determined for each component, depending on whether it is installed locally, installed to run from the source media, such as a CD, or removed.

With InstallShield, you can set your application's disk usage. This allows you to control file costing by choosing to run your application from the source media, from the local machine, or to install it when required. Note that running your application from the source enhances application resiliency and conserves space on an end user's system.

Application Resiliency

If Windows Installer cannot provide a component, the Windows Installer technology enables applications to try to repair the component or to reinstall the component if the corresponding file is corrupted or the current file is older than the available version.

Source Resiliency

In addition to supporting application resiliency, the Windows Installer supports source resiliency through the Source List, which can include network locations, URLs, or compact discs from which applications are installed on-demand. Administrators can use the Group Policy Editor to disable this functionality.

INSTALLDIR vs. TARGETDIR

INSTALLDIR represents the main product installation directory for a Basic MSI and InstallScript MSI installations, such as the end user launching **Setup.exe** or your .msi database.

TARGETDIR represents the installation directory for an InstallScript installation, or for an administrative Windows Installer–based installation (when the user runs **Setup.exe** or **MsiExec.exe** with the `/a` command-line switch).

In an InstallScript MSI project, the InstallScript variable MSI_TARGETDIR stores the target of an administrative installation.

Preventing the Current Installation from Overwriting a Future Major Version of the Same Product



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *MSI Database*
- *Transform*

To prevent end users from being able to install the current version of your product over a future major version of the same product, the Upgrades view should contain a major upgrade item, the major upgrade item should be properly configured to prevent the current version of your product from being installed over a future version, and your project should include a properly configured and scheduled type 19 custom action.

When you create a new Basic MSI or InstallScript MSI project, InstallShield automatically adds support for preventing the current installation from overwriting a future major version:

- The Upgrades view contains a major upgrade item called ISPreventDowngrade.

  The **Products sharing my Upgrade Code** option is selected on the Common tab of the major upgrade item. The value of the Upgrade Code setting on the Advanced tab is {000000000000-0000-0000-0000-00000000}. When you build a release, InstallShield uses the appropriate upgrade code value instead of the placeholder value in the .msi package that it generates at build time. For more information on the upgrade code, see Setting the Upgrade Code.

  InstallShield sets the Detect Property setting of this major upgrade item to ISFOUNDNEWERPRODUCTVERSION and configures the other settings as appropriate.

- The Custom Actions and Sequences view contains a custom action called ISPreventDowngrade, a type 19 custom action that Windows Installer launches when an end user tries to install the current version of your product over a future major version.

  InstallShield schedules the ISPreventDowngrade custom action for the user interface and execute sequences of the installation sequence so that Windows Installer runs it if appropriate, regardless of what user interface level is used. In addition, InstallShield uses ISFOUNDNEWERPRODUCTVERSION as the condition for this custom action.

The following instructions explain how to manually add this support for projects that you created in InstallShield 12 or earlier and then upgraded to InstallShield 2016.

Task ***To manually add support for preventing end users from being able to install the current version of your product over a future major version:***

1. Use the **Upgrades** view to add a major upgrade item to your project.
2. On the **Common** tab, select the **Products using my Upgrade Code** option.
3. Configure the settings on the **Advanced** tab for the major upgrade item as follows:
   a. In the **Minimum Version** setting, specify the product version that you are using for your current project.

      As an alternative, you can enter the following string in this setting:

      *****ALL_VERSION*****

      That value is a placeholder value. If you use that extract string in the **Minimum Version** setting, InstallShield uses the product version of the currently open project instead of the placeholder value in the .msi package that it generates at build time.

   b. Leave the **Maximum Version** setting blank. If a value is listed for this setting, delete it.

   c. If your project includes multiple languages, specify the language identifiers in the **Language** setting.

d. In the **Detect Only** setting, select **Yes**; for all of the other Yes-No settings, select **No**.

e. In the **Detect Property** setting, type a descriptive name such as the following one:

FOUNDNEWERVERSION

4. Add and schedule a type 19 custom action for your project to handle scenarios where an end user tries to install the current version of your product over a future major version:

a. In the **Custom Actions and Sequences** view, right-click the **Custom Actions** explorer and click **New Error**.

b. Select the new custom action.

c. In the **Error Message** setting, type the error message text that should be displayed when an end user tries to install the current version of your product over a future major version.

d. In the **Install UI Sequence** and **Install Exec Sequence** settings, select **After FindRelatedProducts**.

e. In the **Install UI Condition** and **Install Exec Condition** settings, type the value that you specified in step 3e.

Preparing Installations for Non-Administrator Patches

Windows Installer 3.0 and later enables you to create patches that can be installed by non-administrators. Non-administrator patches can be used if strict criteria are met. For example, the base installation that the patch will update must include the certificate that will be used to sign the patch package. To learn about the other criteria that must be met, see Non-Administrator Patches.

Task ***To prepare a base installation that can later be updated by non-administrator patches:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the release whose digital signature information you want to configure.
3. Click the **Signing** tab.
4. Specify the digital signature information.

InstallShield automatically adds the necessary information to the **MsiDigitalCertificate** table and the **MsiPatchCertificate** table. This enables you to create patches that can be installed by non-administrators.

Tip • *The digital signature settings are also available in the Releases view.*

The Windows Installer design enables you to sign a package with one certificate and also allow patches that are signed with a different certificate. The following instructions explain how to add to the base installation the additional certificates for the patches.

Task ***To add additional digital certificates:***

1. Use a tool such as **SignTool.exe** to sign a dummy file. **SignTool.exe** is part of the Windows SDK.
2. In your original installation project, open the **Direct Editor**.

3. In the **Tables** explorer, click **MsiDigitalCertificate**.
4. Click the last row in the table to add a new entry.
5. In the **DigitalCertificate** field, type a unique name for your certificate.
6. Click the **CertData** field. The **Edit Binary Stream** dialog box opens.
7. In the **File name** box, enter the path and name of the file that you signed in step 1. You can click the browse button to find the file.
8. Click **OK**.
9. In the **Tables** explorer, click **MsiPatchCertificate**.
10. Click the last row in the table to add a new entry.
11. In the **PatchCertificate** field, type a unique name for your patch certificate.
12. In the **DigitalCertificate** field, select the entry that you created for the **MsiDigitalCertificate** table in step 4.

Settings for Platforms and Platform Suites



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Note that some of the settings apply to both of these project types, but some apply only to one of these project types.

InstallShield includes several settings for platforms, platform suites, and languages.

Specifying Platforms at the Project Level

One project-level setting enables you to specify the platforms that your project supports:

- **Platform Filtering**—Use this setting to specify the platforms that you want to be available when you select operating system requirements for components or releases in your project. In general, if a platform is not listed for this setting at the project level, you cannot designate that a particular component or release in your project is targeted for that platform.

  The Platforms setting is available for InstallScript projects. To access this setting, open the General Information view. You can also configure this setting through the Project Settings dialog box: on the Project menu, click Settings, and click the Platforms tab.



Note *• Specifying platforms at the project level does not create target system requirements for running the installation. If you want to create target system requirements in an InstallScript project, use the SYSINFO structure to identify the operating platform of the target system.*

For information on how to specify target system requirements for InstallScript MSI projects, see Specifying Operating System Requirements in the Project Assistant.

Specifying Operating Systems and Platform Suites at the Component Level

Two component-level settings enable you to specify platform information for a component:

- **Operating Systems**—If a component is specific to one or more operating systems, use this setting to indicate those operating systems. If the target machine's operating system does not match one of the operating systems that are specified for this setting, the component is not installed.

  By default, components are operating system independent, meaning that none of the component's data are specific to certain operating systems.

  The Operating Systems setting is available for InstallScript and InstallScript MSI projects. To access this setting, open the Components view and select a component.

- **Platform Suite(s)**—If a component is specific to one or more platform suites, use this setting to indicate the suites. If you specify more than suite, you can indicate whether all or any of the suites must be present on the target machine in order for the component to be installed.

  By default, components are platform suite independent, meaning that none of the component's data are specific to a particular platform suite.

  This setting provides an additional layer of filtering beyond the Operating Systems setting. Set the Platform Suite(s) setting only if necessary, and be sure to select only those platform suites that are required for the proper functioning of your application. For example, if a component should be installed on both the Home and Professional editions of Windows XP, you can leave Suite Independent as the value for this setting; selecting Windows XP for the Operating Systems setting encompasses both editions.

  The Platform Suite(s) setting is available for InstallScript projects. To access this setting, open the Components view and select a component.

Specifying Platforms at the Release Level

One release-level setting enable you to specify platform information for a release:

- **Platform(s)**—If a release is specific to one or more platforms, use this setting to indicate the platforms. If the platform specified for a component does not match one of the platforms that is selected for this setting, InstallShield does not include the component in the release.

  The default value for this setting is Use Project Setting. This value indicates that the release supports the platforms that are specified at the project level.

  The Platform(s) setting is available for InstallScript projects. To access this setting, open the Releases view and select a release. You can also configure this setting through the Platforms panel in the Release Wizard.

Controlling Support for Platforms and Platform Suites at Run Time for InstallScript Projects

During run time of InstallScript installations, you can control the platforms and platform suites that your installation supports by calling the **FeatureFilterOS** function.

In the **OnFilterComponents** event handler, the framework typically calls this function with the platforms and platform suites that match the target system so that only the appropriate components are installed. By calling **FeatureFilterOS**, you can override this default behavior to install or prevent the installation of components based on any platform or platform suite criteria that you specify.

For more information, see FeatureFilterOS.

Specifying Installation Information

As you begin creating your installation project, you will need to specify important installation information at the outset. This includes specifying product and project settings and configuring Add or Remove Programs settings.

Configuring General Project Settings

InstallShield stores your project settings in a single installation project file (.ism file). This file stores all of the information about your project. In the General Information view, you can edit basic information about your installation project—including the author's name, the languages that the project supports, and any comments that you want to include.

The General Information view is also where you configure general product information such as the product name, the product code (GUID), and the version number. A product is the top level of organization in an installation project. The installation is further divided into features and components, which are subsets of your product. Although an installation can contain multiple features and components, it can have only one product.

For detailed information about each of the project and projects settings that are available, see General Information View.

Saving an InstallShield Project File (.ism) in XML or Binary Format

InstallShield lets you save your .ism project file in XML or binary format.



Task ***To specify the format of your project file:***

1. In the View List under **Installation Information**, click **General Information**.
2. In **Project File Format** setting, select the appropriate option. Available options are:
   - **Binary**—To save the project file as a database file, select this option. This format is best for the speed of opening and saving the project file.
   - **XML**—To save the project file as a hierarchical text-based format, select this option. This project file format is best for use with source code control systems.



Note • *Your InstallShield project file (.ism) retains the .ism file extension when you save it in XML or binary format.*

Advanced UI and Suite/Advanced UI projects (.issuite) are always saved as XML files.

Specifying a Product Name

Enter the name of your product in the Product Name setting of the General Information view. The name that you enter should be the name of the product for which you are creating an installation. This value is used throughout your project, in various instances; for example:

- The name of the source file folder under the project location

- In Basic MSI and InstallScript MSI projects: the name of the Windows Installer package (.msi file) that InstallShield builds by default
- In run-time dialogs
- Under the informational registry key in accordance with Windows logo requirements. The informational values are found in the following location and are used in Add or Remove Programs to enable an end user to change or remove your product.

  HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall*ProductCode*

Since it will be incorporated into the paths for your source files, the product name cannot contain any of the following characters: \ / : * ? " < > | . -



Tip *• In an InstallScript or InstallScript MSI project, you can use the UNINSTALL_DISPLAYNAME variable to specify a different product display name in Add or Remove Programs.*

For Basic MSI and InstallScript MSI projects—If you do not want to use the product name for the .msi package file name, you can use the MSI Package File Name setting on the General tab for a product configuration in the Releases view and specify a different .msi file name. Note that if you want to be able to release minor upgrades or small updates to update your product, the previous and latest versions of your installation must have the same .msi package name. Attempting to perform a minor upgrade or a small update when the .msi file name has changed can lead to Windows Installer run-time error 1316.



Caution *• If you want to include an ampersand (&) in your product name, you must use two ampersands (&&) to display the name properly in end user dialogs. For example, to display* ***New & Improved Product****, you should enter the product name as* **New && Improved Product**.

Specifying the Product Version

When you are specifying the version number for your product, you must ensure that you enter a valid product version. The version must contain only numbers. It is typically in the format *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535.

At run time, the installation registers the version number of the product that is being installed. The entire version string is displayed in Add or Remove Programs. The product version number is important because the installation engine uses it in part to determine whether to apply an upgrade.

To configure the product version, use the General Information view. This product version can be overridden in other areas of InstallShield—for example, for a product configuration in the Releases view of a Basic MSI project. You can also specify the version number that you want an upgrade to target in other areas of InstallShield—for example, in the Upgrades view of a Basic MSI or InstallScript MSI project.

Note that although you can include the fourth field (*ddddd*) when you specify your product's version, the installation does not use this part of the product version to distinguish between different product versions.

Project *• For Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects—If your release includes a* **Setup.exe** *file, the product version that you specify is displayed on the Properties dialog box for* **Setup.exe***. For more information, see Customizing File Properties for the Setup Launcher.*

For InstallScript and InstallScript Object projects—Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value. (To use a path variable: On the Project menu, click Settings. Then select the appropriate path variable on the Application tab.)

Product Version Numbers in InstallScript and InstallScript Object Projects

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

By default, InstallScript projects use version numbers in packed DWORD format—that is, as a four-byte value whose first byte is the major version, second byte is the minor version, and last two bytes are the build number. Packed DWORDS are entered and displayed in the format major.minor.build; for example, 1.2.3 or 255.255.65535. Packed DWORD version numbers are assumed in the default script code for registering the version number of the product that is being installed, and for comparing that version number to that of the already installed product during an update installation.

If any of these version numbers are not in packed DWORD format, you must modify the script code as discussed in the following sections.

Version Number of the Already Installed Product

The default **OnSetUpdateMode** event handler function code compares the version number of the product that is being installed, and the version numbers to which the update can be applied (as specified in the Release property sheet), to the system variable *IFX_INSTALLED_VERSION*. The installation automatically attempts to initialize the value of this system variable to the string equivalent of the data in the Version value under the application uninstallation registry key. If that value does not exist, or its data is not a packed DWORD, then the value of *IFX_INSTALLED_VERSION* is a null string (""), in which case the default **OnSetUpdateMode** code displays an error message and aborts the installation. One solution is to insert code that checks the version information on the system and sets *IFX_INSTALLED_VERSION* equal to an appropriate packed DWORD value. For example, if previous installations stored a version string in the MyVersion value under HKEY_LOCAL_MACHINE\SOFTWARE\MyCompany\MyProduct, you could insert code like the following:

```
if (IFX_INSTALLED_VERSION="") then
    /* Get the registered version information. */
    RegDBSetDefaultRoot( HKEY_LOCAL_MACHINE );
    RegDBGetKeyValueEx( "Software\\MyCompany\\MyProduct",
        "MyVersion", REGDB_STRING, szVersionString, nSize );

    /* Assign a value to IFX_INSTALLED_VERSION. */
    switch (szVersionString)
        /* A registered version string of "A" corresponds to an existing version number of 1.0.0. */
        case "A":
```

```
            IFX_INSTALLED_VERSION = "1.0.0";
        /* A registered version string of "B" corresponds to an existing version number of 1.1.0. */
        case "B":
            IFX_INSTALLED_VERSION = "1.1.0";
        /* An absent version string corresponds to an existing version number of 0.0.0. */
        default:
            IFX_INSTALLED_VERSION = "0.0.0";
    endswitch;
endif;
```

Version Number of the Product that Is Being Installed

The default **OnMoveData** event handler function code calls **RegDBSetVersion** to take the product version number that you entered in the General Information view and enter it in the target system's registry. **RegDBSetVersion** assumes that the product version is in packed DWORD format. If you want to register a product version that is not in packed DWORD format, you must replace OnMoveData's call of **RegDBSetVersion** with code like the following:

```
/* The setup automatically initializes the value of the system variable
IFX_PRODUCT_VERSION to the product version that you entered in the
Product Version setting in the General Information view. */
RegDBSetItem(REGDB_UNINSTALL_VERSION, IFX_PRODUCT_VERSION );
RegDBSetItem(REGDB_UNINSTALL_DISPLAY_VERSION, IFX_PRODUCT_VERSION );
```

RegDBSetItem must be called after **MaintenanceStart**.

Setting the Product Code in a Windows Installer–Based Project



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The product code is a string that uniquely identifies your product. An installation uses a product code at run time to determine whether the product has already been installed.

Enter a GUID that uniquely identifies your product, or click the **Generate a new GUID** button ({...}) in the Product Code setting in the General Information view to let InstallShield generate a new GUID. The installation registers this GUID at run time.



Caution • *Because the product code uniquely identifies your product, changing the code after distributing a release of your product is not recommended.*

Setting the Product Code in an InstallScript-Based Project



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The product code is a GUID string that uniquely identifies your product. An installation uses a product code at run time to determine whether the product has already been installed.

Enter a GUID that uniquely identifies your product, or click the **Generate a new GUID** button ({...}) in the Product Code setting in the General Information view to let InstallShield generate a new GUID. The installation registers this GUID at run time.



Caution *• The project GUID is used to associate uninstallation or maintenance with the original installation. A new GUID is automatically generated for each new project that you create, including copies of existing projects. Once you have changed a project's GUID, its previous GUID cannot be recovered. For these reasons, changing a project's GUID is typically not necessary and should be approached with caution.*

Setting the Upgrade Code



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *QuickPatch*
- *Transform*

The upgrade code is a GUID that uniquely identifies the product family to which the product belongs. In most cases, the upgrade code should be consistent across different versions and languages of a family of related products so that Windows Installer can use it to search for related versions of the product that are already installed.

To configure the GUID for a new product, use the Upgrade Code setting in the General Information view. This upgrade code value can be overridden in other areas of InstallShield—for example, for a product configuration in the Releases view, or for each instance that is defined on the Multiple Instances tab for a product configuration in the Releases view.

When you are working on an upgrade for a new version of your product, use the upgrade code that was configured in the General Information view (or overridden in one of the aforementioned other areas of InstallShield) as the GUID that you specify in the Upgrades view of your upgrade project or in your QuickPatch project; when these upgrade codes match, the upgrade or patch can target the base version, and Windows Installer can update it as needed.

Setting Product Conditions



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

- *Transform*

The Installation Requirements page in the Project Assistant lets you specify some commonly used installation requirements for the target system. For example, if your application requires a specific operating system in order to run properly, you can indicate that on the Installation Requirements page.

InstallShield also enables you to define your own custom conditions that Windows Installer must evaluate before your product can be installed. Conditions that you create in the General Information view and on the Installation Requirements page in the Project Assistant apply to the entire product; if one or more of the conditions are false on the target system, the installation exits and an error message is displayed.

For example, if your product requires 64 MB of RAM in order to run properly, you can use the Product Condition Builder dialog box to create a condition. At run time, the Windows Installer checks the target system to determine how much RAM is installed. If it is less than 64 MB, or if any of the other product conditions are false, the Windows Installer displays an error message and exits the installation.



***Tip** • You cannot guarantee the order in which Windows Installer evaluates product launch conditions. If it is necessary to control the order in which the conditions are evaluated, you can create an error custom action in the Custom Actions and Sequences view for each condition, and schedule them in the appropriate order.*



Task ***To create a custom launch condition for your product's installation:***

1. In the View List under **Installation Information**, click **General Information**.
2. Click the **Install Condition** setting and then click the ellipsis button (**...**). The **Product Condition Builder** dialog box opens.
3. Click the **New Condition** button. InstallShield adds a new condition row to the **Conditions** box.
4. Do one of the following:
   - In the **Condition** column, type the launch condition.
   - Use the **Properties** list, the **Operators** list, and the **Add** buttons to build your conditional statement:
     a. In the **Properties** list, select a property and then click the **Add** button. InstallShield adds the property to the **Condition** column.
     b. If your conditional statement should contain an operator, select an operator in the **Operators** list and then click the **Add** button. InstallShield adds the operator to the conditional statement.
     c. If your conditional statement should contain a value, double-click the condition field, press END so that the insertion point is at the end of the condition statement, and enter the value.
5. In the **Message** column, enter the error message that you would like to be displayed if the condition evaluates to false.

   When you type a value for this column, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can double-click this setting and then click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.



Note • *Windows Installer dialogs, which display the text that you specify in the Message column, do not recognize the escape sequences \r (carriage return), \n (new line), or \t (tab).*

6. Click **OK**.



Important • *InstallShield performs basic condition validation; however, you should still double-check that your condition statements evaluate to the expected outcome. For more information and example conditions, see Building Conditional Statements.*

Setting the Default Product Destination Folder (INSTALLDIR)



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Your project's **INSTALLDIR** property serves as the default folder for all of your product's files. Its value is assigned to the Windows Installer folder property **INSTALLDIR**, which is the default feature and component destination folder.



Windows Logo • *According to the Windows logo program requirements, the default destination of your product's files must be a subfolder of the Program Files folder or the end user's Application Data folder, regardless of the language of the target system. If you use ProgramFilesFolder as the parent folder for your product's destination folder setting, your files are installed to the correct location. The default value for the INSTALLDIR setting is:*

[ProgramFilesFolder]*Company Name******Product Name*



Task ***To set a product's INSTALLDIR property:***

1. In the View List under **Installation Information**, click **General Information**.
2. To use a built-in Windows Installer directory as part of your path: In the **INSTALLDIR** setting, click the ellipsis button (...) . The **Set INSTALLDIR** dialog box opens. In the **Destination Directories** box, select a destination folder.

   As an alternative, you can manually enter the path in the **INSTALLDIR** setting.



Note • *Selecting a new folder property in the Set INSTALLDIR dialog box overwrites the contents of the value in the INSTALLDIR setting. You can specify a subfolder of any folder property by separating subfolders with a backslash—for example,* ***[ProgramFilesFolder]My Company\Program****.*

Other Destination Folder Considerations (Windows Installer–based projects only)

Setting the feature's Remote Installation setting to Favor Source—or to Favor Parent when the subfeature's parent feature is set to Favor Source—means that the feature's files will not be installed on the target system, regardless of the product's **INSTALLDIR** property.

Each feature and component has a Destination setting. The feature's Destination setting overrides the product's Destination Folder setting, and the component's Destination setting overrides the feature's. Therefore, if you want all of your product's files to be installed by default to the product's destination folder, enter **[INSTALLDIR]** for all of your features' and components' Destination settings.

When using an installer folder property such as **INSTALLDIR**, you are specifying a *default* value. An end user could change this value by setting a property when launching **Msiexec.exe** at the command line or by selecting a new destination folder for a feature in the CustomSetup dialog.

Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment

InstallShield offers several ways to secure files, folders, registry keys, and Windows services for end users who run your product in a locked-down environment:

- **Traditional Windows Installer handling**—In Windows Installer–based projects, you can choose to use the built-in Windows Installer support for setting permissions for files, folders, and registry keys at run time. With this option, InstallShield stores permission information for your product in the **LockPermissions** table of the .msi database.

  This type of permission handling cannot be combined with the new Windows Installer handling; if you try to build a release that contains the **MsiLockPermissionsEx** table and the **LockPermissions** table, build error -7207 occurs.

- **New Windows Installer handling**—In Windows Installer–based projects, you can choose to use the latest Windows Installer support for setting permissions for files, folders, registry keys, and Windows services at run time. With this option, InstallShield stores permission information for your product in the **MsiLockPermissionsEx** table of the .msi database.

  This option requires Windows Installer 5 or later on the target system; earlier versions of Windows Installer ignore settings for this type of handling.

  This type of permission handling cannot be combined with the traditional Windows Installer handling; if you try to build a release that contains the **MsiLockPermissionsEx** table and the **LockPermissions** table, build error -7207 occurs.

- **Custom InstallShield handling**—In Windows Installer–based projects, you can choose to use custom support for setting permissions at run time. With this option, InstallShield stores permission information for your product in the custom **ISLockPermissions** table of the .msi database. InstallShield also adds custom actions to your project.

- **SetObjectPermissions, an InstallScript Function**—You can use the **SetObjectPermissions** function in InstallScript events and InstallScript custom actions to set permissions at run time.

All of these methods enable you to assign permissions for a file, folder, or registry key to specific groups and users. For example, you may assign Read, Write, and Delete permissions for a particular file to the Administrators group, but only Read permissions for all of the users in a different group. The new Windows Installer handling option also lets you assign permissions for a Windows service.

Determining Which Option to Use

The following table compares the different types of methods for setting permissions.

Table 4-1 • Comparison of Different Ways to Secure Objects in a Locked-Down Environment

Comparison Category	Explanation of Available Support
Project type	• **Traditional Windows Installer handling, New Windows Installer handling, and Custom InstallShield handling**—Available in the following project types: Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform. • **SetObjectPermissions function**—Available in InstallScript events in the following project types: InstallScript, InstallScript MSI. Also available through InstallScript custom actions in the following project types: Basic MSI, DIM, InstallScript MSI, and Merge Module.
Well-known security identifiers (SIDs)	• **Traditional Windows Installer handling**—Supports a limited number of SIDs (Administrators, Everyone). • **New Windows Installer handling, Custom InstallShield handling, SetObjectPermissions function**—Supports many SIDs (Administrators, Authenticated Users, Creator Owner, Everyone, Guests, Interactive, Local Service, Local System, Network Service, Power Users, Remote Desktop Users, and Users).
Localized names for SIDs	• **Traditional Windows Installer handling**—Does not support localized names for SIDs; if you try to use a localized name, the installation fails. • **New Windows Installer handling, Custom InstallShield handling, and SetObjectPermissions function**—Supports localized names for all of the supported well-known SIDs (Administrators, Authenticated Users, Creator Owner, Everyone, Guests, Interactive, Local Service, Local System, Network Service, Power Users, Remote Desktop Users, and Users).
Ability to deny specific permissions	• **Traditional Windows Installer handling**—Not supported. This handling lets you set specific permissions; you cannot deny permissions. Thus, you can give a user read-only access to a file. However, you cannot prevent a user from having read-only access. • **New Windows Installer handling, Custom InstallShield handling, and SetObjectPermissions function**—Supported. These options let you indicate whether you want to deny a user or group from having the permissions that you are specifying.

Table 4-1 • Comparison of Different Ways to Secure Objects in a Locked-Down Environment (cont.)

Comparison Category	Explanation of Available Support
Effect on permissions that already exist	• **Traditional Windows Installer handling**—Existing permissions may be deleted. For example, if permissions are already set for a folder on the target system for the Everyone user, and your installation needs to set permissions for the Administrators user, this option would allow you to set permissions for the Administrators user. However, the existing permissions for Everyone would be deleted. • **New Windows Installer handling, Custom InstallShield handling, and SetObjectPermissions function**—These options let you add permissions to a file, folder, or registry key that already exists on the target system, without deleting any existing permissions for that object. For example, if permissions are already set for a folder on the target system for the Everyone user, and your installation needs to set permissions for the Administrators user, these options would allow you to set permissions for the Administrators user without deleting the existing permissions for the Everyone user.
Ability to propagate permissions to child objects (subfolders, files, and subkeys)	• **Traditional Windows Installer handling**—Not supported. If you want to configure permissions for a subfolder or a file in a folder (or a subkey under a registry key), the parent that is created on the target system automatically inherits the permissions of its child. • **New Windows Installer handling, Custom InstallShield handling, and SetObjectPermissions function**—Supported. These options let you configure permissions for a folder (or a registry key), and indicate whether you want the permissions to be applied to all of the folder's subfolders and files (or the registry key's subkeys).
Ability to set permissions for objects that are not being installed as part of your installation	• **Traditional Windows Installer handling, New Windows Installer handling, and Custom InstallShield handling**—Not supported. • **SetObjectPermissions function**—Supported. You can secure permissions for a file, folder, or registry key that is installed as part of your installation, or it can be already present on the target system.
Ability to set permissions for a new user that is being created during the installation	• **Traditional Windows Installer handling**—Not supported. • **New Windows Installer handling, Custom InstallShield handling, and SetObjectPermissions function**—Supported. If a new user is created during the installation, you can configure permissions for that user.

Learning More about the Custom InstallShield Handling Option or the Traditional Windows Installer Handling Option

In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects, you need to specify whether you want to use the custom InstallShield handling or the Windows Installer handling. To learn how, see Selecting the Locked-Down Permissions Type for a Project.

To learn how to set permissions for a file or folder using either of these options, see Configuring Permissions for Files and Folders. For information on setting permissions for a registry key using either of these options, see Configuring Permissions for Registry Keys.

Learning More about the New Windows Installer Handling Option

To use the new Windows Installer handling option for a service, add a service to your project and then configure its settings. For more information, see Installing, Controlling, and Configuring Windows Services.

To use the new Windows Installer handling option for files, folders, or registry keys, use the **MsiLockPermissionsEx** table in the Direct Editor view.

Learning More about the InstallScript Function SetObjectPermissions

For information on the **SetObjectPermissions** function, see SetObjectPermissions.

Selecting the Locked-Down Permissions Type for a Project



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield includes a project-wide setting that lets you specify how your installation should configure permissions for files, folders, and registry keys for end users in a locked-down environment.



Task ***Selecting the locked-down permission type for a project:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Locked-Down Permissions** setting, select the appropriate option:
   - **Custom InstallShield handling**—InstallShield adds a custom table and custom actions to your project to set permissions on the target system. This is the default value.
   - **Traditional Windows Installer handling**—InstallShield uses the **LockPermissions** table in the .msi database to store permissions information for your product.

For a detailed comparison of these two options, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.

The next time that you configure permissions for a file, folder, or registry key in your project, InstallShield uses the locked-down permission type that you selected:

- If you selected the traditional Windows Installer handling option, InstallShield uses the **LockPermissions** table in your project.

- If you selected the custom InstallShield handling option, InstallShield uses the **ISLockPermissions** table in your project; InstallShield also adds the ISLockPermissionsCost and ISLockPermissionsInstall custom actions to your project.

If you change the value of the Locked-Down Permissions setting but your project already contains permission settings for files, folders, or registry keys, InstallShield displays a message box that lets you specify whether you want to migrate the permission data to the appropriate table. If you choose to migrate the data, InstallShield moves the data to the table that corresponds with the option that you selected; if you are switching from the custom InstallShield handling option to the traditional Windows Installer handling option, InstallShield also deletes the ISLockPermissionsCost and ISLockPermissionsInstall custom actions from your project.

Specifying Whether Windows Installer Installations Should Be Logged

InstallShield enables you to specify on a project-wide basis whether Windows Installer 4.0 or later should log your installation. You can also customize the types of messages that are logged.



Task ***To specify project-wide logging information for Windows Installer 4.0 or later:***

1. In the View List under **Installation Information**, click **General Information**.
2. Click the **Create MSI Logs** setting, and then click the ellipsis button (**...**). The **Logging Options for Windows Installer 4.0 and Later** dialog box opens.
3. Select the appropriate option. If you select the custom option, enter the MsiLogging value.

   For a list of valid parameters for the MsiLogging value, see MsiLogging Property in the Windows Installer Help Library.
4. Click **OK**.

The results vary, depending on which option you select in the Logging Options for Windows Installer 4.0 and Later dialog box:

- If you select No, installations are not logged. This is the default value.
- If you select Yes, InstallShield populates the **MsiLogging** property with the default value of *voicewarmupx*. If the installation is run on a target system that has Windows Installer 4.0, the following occurs:
  - The installer creates a log file according to the default logging mode of *voicewarmupx*.
  - The installer populates the **MsiLogFileLocation** property with the log file's path.
  - A **Show the Windows Installer log** check box is added to the SetupCompleteSuccess, SetupCompleteError, and SetupInterrupted dialogs. If the end user selects that check box and then clicks Finish, the log file is opened in a text file viewer or editor.
- If you select Custom, InstallShield populates the **MsiLogging** property with the value that you specify in the box. If the installation is run on a target system that has Windows Installer 4.0, the following occurs:
  - The installer creates a log file according to the custom value that you specified in the box.
  - The installer populates the **MsiLogFileLocation** property with the log file's path.

- A **Show the Windows Installer log** check box is added to the SetupCompleteSuccess, SetupCompleteError, and SetupInterrupted dialogs. If the end user selects that check box and then clicks Finish, the log file is opened in a text file viewer or editor.

Earlier versions of Windows Installer ignore the MsiLogging setting. The **Show the Windows Installer log** check box is not visible in run-time dialogs that are displayed on systems running earlier versions of Windows Installer.



Important • *The **MsiLogFileLocation** property is read-only; it cannot be used to set or change the log file location.*

Running an InstallScript Installation Multiple Times

The InstallScript project type offers several options for the behavior of your installation when a target machine already has your product installed and the end user reruns the installation.

Table 4-2 • Maintenance Experience Settings

Maintenance Experience	When Maintenance UI Is Displayed	Number of Entries Created in Add or Remove Programs
Standard	Whenever the installation is rerun	One
Multi-Instance	Only when the installation is rerun from the Add or Remove Programs	Multiple—a separate entry for each instance
No uninstall or maintenance	Never	None

You can select these options from the Maintenance Experience setting in the General Information view.

For the default behavior—the standard option—the maintenance user interface is displayed if end users rerun an installation on machines.

The multi-instance option lets your end users rerun an installation multiple times as a first-time installation rather than as a maintenance installation. By default, this option lets end users install the product to a different location each time that they run the installation. Each time that they run the installation as a first-time installation, the components that they select (whether directly or by selecting a setup type) are installed.

If one or more instances of your product are present on a machine when an end user reruns the multi-instance installation with a user interface, the installation displays the Qualifying Product(s) Detected dialog. This dialog enables the end user to select the instance to which the update should be applied. However, if the end user reruns the installation silently, the installation suppresses the Qualifying Product(s) Detected dialog and creates a new instance on the machine.



Project • *For information on multiple-instance support in Basic MSI projects, see Installing Multiple Instances of Products.*

Configuring the Enable Maintenance Setting

Project • *This information applies to InstallScript MSI projects.*

Use the Enable Maintenance setting in the General Information view to indicate whether you want to display the full maintenance user interface (UI) or the uninstallation UI when end users rerun the installation on a system on which the product is already present.

The following table shows the available options for the Enable Maintenance setting.

Table 4-3 • Enable Maintenance Setting Options

Option	Description
Yes	Unless the `/removeonly` command-line parameter is passed to **Setup.exe**, the system variable REMOVEONLY is set to FALSE when an end user reruns the installation, and the standard maintenance UI is displayed.
No	The system variable REMOVEONLY is set to TRUE when an end user reruns the installation, and the uninstallation UI is displayed. ***Note*** • *Selecting No for the Enable Maintenance setting does not cause the OnUninstall event handler to be called. If you want the OnUninstall event handler to be called, you must use the* `/uninst` *command-line parameter for* **Setup.exe**. *Note that this is applicable only if the InstallScript UI style is the traditional style, which uses the InstallScript engine as an external UI handler. The* `/uninst` *command-line parameter is not supported if the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.*

Note • *InstallScript MSI installations show separate Change and Remove buttons in Add or Remove Programs in the Control Panel by default. Clicking the Change button always displays the maintenance UI, and clicking the Remove button always displays the uninstallation UI.*

The Enable Maintenance setting has no effect on the behavior that occurs if end users initiate maintenance for your product by clicking the Change button for your product's entry in Add or Remove Programs.

If you want to prevent end users from being able to run maintenance from within Add or Remove Programs, select Yes for the Disable Change Button setting in the General Information view.

Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations

Project • *This information applies to InstallScript MSI projects.*

InstallScript MSI installations use two different installer engines:

- The Windows Installer engine runs the standard Execute sequence of the .msi package. This is the sequence that typically modifies the target system.
- The InstallScript engine serves as the custom user interface (UI) handler of the installation. It also executes the InstallScript code. The advantage of using the InstallScript engine for the UI is that it offers support for highly customized run-time dialogs.

Traditionally, Windows Installer has had support for only external custom UI handlers; therefore, InstallScript MSI installations have always required a **Setup.exe** setup launcher. The setup launcher serves as a bootstrap application that initiates the InstallScript engine to display the UI and run the InstallScript code, and the Windows Installer to run the Execute sequence of the .msi package.

Windows Installer 4.5 introduces support for embedded custom UI handlers. If the InstallScript engine is embedded within the .msi package, an InstallScript MSI installation does not require the **Setup.exe** setup launcher; end users can launch the installation by launching the .msi package. In this case, the .msi package contains the information that the Windows Installer needs to know about launching the InstallScript engine for the installation's UI.

Thus, InstallShield offers two different styles for the UI of InstallScript MSI installations:

- Traditional style (requires a **Setup.exe** setup launcher)—This style lets you use the InstallScript engine as an external UI handler for your InstallScript MSI installation. This is the default style.
- New style (requires Windows Installer 4.5 on the target machine)—This style lets you use the InstallScript engine as an embedded UI handler for your InstallScript MSI installation. With this style, InstallShield embeds the InstallScript engine within the .msi package. Note that this option has some limitations.

The following sections provide detailed information about these two styles; review this information to determine which style would best meet your requirements.

Traditional Style (InstallScript Engine as an External UI Handler)

The general flow in a traditional-style InstallScript MSI package during a first-time installation is as follows:

1. The InstallScript engine opens the .msi package.
2. All actions in the Installation UI sequence are run with the exception of the ExecuteAction. The actions are run in ascending sequence order.
3. The InstallScript engine performs its own internal component costing to determine the space required for all features and components in the installation.
4. The InstallScript engine launches the compiled script contained in **ISSetup.dll**. The event sequence for the events that are run before the .msi package installation is as follows:
   a. OnBegin
   b. On*Xxx*UIBefore (where **Xxx** is First, Maint, Admin, Patch, Resume, etc.)
   c. OnGeneratingMsiScript
   d. OnMoving
   e. OnFeaturesInstalling (Any feature installing and uninstalling events are run at this point.)
   f. OnInstallFilesActionBefore
   g. OnGeneratedMsiScript

5. The InstallScript engine launches the .msi package installation through **MsiInstallProduct**. The following factors determine the command line that the InstallScript engine passes through **MsiInstallProduct**: installation mode (for example, first-time installation, maintenance mode, minor upgrade, patch), internal feature selections, and current property values.
6. If the .msi package installation is successful, the InstallScript engine writes the secondary uninstall key (InstallShield_{ProductCode}) to the machine and then launches the following events:
   a. OnInstallFilesActionAfter
   b. OnFeaturesInstalled (Any feature installed and uninstalled events are run at this point.)
   c. OnMoved
   d. On*Xxx*UIAfter (where *Xxx* is First, Maint, Admin, Patch, Resume, etc.)
   e. OnEnd
7. After the script has completed, the InstallScript engine writes the InstallScript log to the machine.
8. The InstallScript engine shuts down.

New Style (InstallScript Engine as an Embedded UI Handler)

The InstallScript MSI embedded UI support attempts to follow the same general flow as traditional-style InstallScript MSI installation. However, there are certain portions of the flow that do not occur or are unnecessary. The general flow in a new-style InstallScript MSI package during a first-time installation is as follows:

1. The .msi package is launched directly or by **Setup.exe** (which then launches **Msiexec.exe**, which is the same behavior for a Basic MSI installation).
2. The Windows Installer engine extracts all files that are in the **MsiEmbeddedUI** table and calls the initialize embedded UI function in the DLL that contains the embedded UI handler attribute. (In the InstallScript MSI case, this is **ISSetup.dll**).
3. The InstallScript engine initializes and manually launches the ISSetupFilesExtract action if it is present and its condition in the InstallUISequence evaluates to true.
4. The InstallScript engine performs manual resolution of all entries in the **Directory** table.
5. The InstallScript engine performs its own internal component costing to determine the space required for all features and components in the installation.
6. The InstallScript engine launches the compiled script contained in **ISSetup.dll**. The event sequence for the events that are run before the .msi package installation is as follows:
   a. OnBegin
   b. On*Xxx*UIBefore (where *Xxx* is First, Maint, Admin, Patch, Resume, etc.)
   c. OnGeneratingMsiScript
   d. OnMoving
   e. OnFeaturesInstalling (Any feature installing and uninstalling events are run at this point.)
   f. OnInstallFilesActionBefore
   g. OnGeneratedMsiScript

7. The InstallScript engine returns control to the Windows Installer engine, which then begins running the installation's Execute sequence.
8. If the .msi package installation is successful, the Windows Installer engine calls back into **ISSetup.dll**. **ISSetup.dll** then launches the following events:
   a. OnInstallFilesActionAfter
   b. OnFeaturesInstalled (Any feature installed and uninstalled events are run at this point.)
   c. OnMoved
   d. On*Xxx*UIAfter (where *Xxx* is First, Maint, Admin, Patch, Resume, etc.)
   e. OnEnd
9. The InstallScript engine shuts down and returns control to the Windows Installer.

For the new-style InstallScript MSI installations, the Install UI sequence is not run. If any custom actions are sequenced in the Install UI sequence, they will not run, similar to how such actions would not run in a Basic MSI that is launched with `/qb` or `/qn`.

The InstallScript engine does not log any changes that are made to the system from InstallScript events.

Most command-line parameters that are supported by traditional-style InstallScript MSI installations are also supported for new-style InstallScript MSI installations. If an end user launches the .msi package directly, these parameters can be passed through the ISSCRIPTCMDLINE property.

Limitations with the Traditional Style (InstallScript Engine as an External UI Handler)

Uninstallable Patch Support

The traditional style does not have support for creating uninstallable patches.

Limitations and Known Issues with the New Style (InstallScript Engine as an Embedded UI Handler)

If you are considering the new style, note the following details.

Windows Installer 4.5 Requirement

The new style requires that Windows Installer 4.5 be present on the target system. You can add InstallShield prerequisites for Windows Installer 4.5 to your project so that Windows Installer 4.5 is installed if it is not present. If you include these InstallShield prerequisites in your installation, you need to use a **Setup.exe** setup launcher. For more information, see Adding Windows Installer Redistributables to Projects. Note that Windows Installer 4.5 cannot be installed on some of the early versions of Windows.

If Windows Installer 4.5 is not installed on a target system at run time, and the installation is launched with Setup.exe, the setup launcher displays error 1713 and indicates that the installation requires Windows Installer 4.5 or later in order to run. The installation aborts after the end user dismisses this error message. If the .msi package is launched directly, the installation displays an error indicating that the package needs to be launched from the **Setup.exe** file.

Upgrade and Patch Support

InstallShield does not include any built-in support for creating an upgrade that uses the new style if the upgrade updates a product that was installed with the traditional style. Therefore, if you use the traditional style to build a release of your product and then you later create an upgrade, consider using the traditional style again for the upgrade. This applies to all types of upgrades (major upgrades, minor upgrades, and small updates) that are packaged as full-installations or as patches.

Note that if the previous setups that a patch is targeting use the traditional style, the patch should also use the traditional style. Also note that a QuickPatch project maintains the UI style from the original installation. Therefore, if the original installation used the traditional style, the QuickPatch package also uses the traditional style. If the original installation used the new style, the QuickPatch package also uses the new style.

As a workaround for the major upgrade scenario, you could consider having your new-style major-upgrade installation read the uninstall string of the earlier product from the registry, and launching it with the InstallScript function LaunchApplication to uninstall the earlier version of your product before installing the new version. Note that you may need to set LAAW_SHELLEXECUTEVERB to **runas** before using **LaunchApplication** in your script.

MsiDoAction

Attempting to call the **MsiDoAction** function from one of the events that are run before the .msi package installation returns an error. This error occurs because of the handle that the Windows Installer passes to the InstallScript engine. The handle does not support running standard or custom actions in the .msi package. This function can be called from InstallScript custom actions if needed.

MsiGetTargetPath and MsiSetTargetPath

The Windows Installer **MsiGetTargetPath** and **MsiSetTargetPath** functions do not work correctly in the new style if they are in event-driven script. These functions rely on the Directory Manager having been initialized by the Windows Installer costing actions. In new-style InstallScript MSI installations, Windows Installer costing is not performed. Thus, **MsiGetTargetPath** fails to return any path information, and **MsiSetTargetPath** fails to set the requested target path. In both cases, Windows Installer logs messages in a verbose log if either of these functions are called.

To update installation paths, use FeatureSetTarget.

Note that **MsiGetTargetPath** and **MsiSetTargetPath** can be called through InstallScript custom actions that are sequenced after CostFinalize in the Installation Execute sequence.

FeatureTransferData and ComponentTransferData

The behavior of the InstallScript functions **FeatureTransferData** and **ComponentTransferData** is undefined for the new style of InstallScript MSI installations and should not be used.

program...endprogram

New-style InstallScript MSI installations cannot use procedural scripts (that is, those with a program...endprogram block). The program function is not called in this case. An event-driven script should be used to customize the behavior of the installation.

/uninst

New-style InstallScript MSI installations do not support the `/uninst` command-line parameter.

BATCH_INSTALL

Attempting to reboot by setting the BATCH_INSTALL variable does not work correctly. To perform a reboot, use the ScheduleReboot action or the REBOOT property.

Reboot Support

Reboots for new-style InstallScript MSI installations are handled in the same manner as they are handled for Basic MSI installations—not as they are handled for InstallScript-based installations (where the installation resumes after the restart and the **OnRebooted** event handler is called).

New-style InstallScript MSI installations should be authored similarly to Basic MSI installations to handle reboots; the installation does not run after a scheduled reboot occurs, and the **OnRebooted** event handler is not called.

Msiexec.exe /x (for Uninstallation)

If an end user tries to uninstall the product through the command line using the statement `Msiexec.exe /x {ProductCode}`, the uninstallation may be successful. However, the Windows Installer displays an error dialog near the end of the uninstallation. The error dialog indicates that the Windows Installer service could not be accessed.

To perform an uninstallation from the command line, the current recommended method is to use one of the following:

```
msiexec.exe /i {ProductCode} REMOVE=ALL
```

```
msiexec.exe /x {ProductCode} /qn
```

Neither of these examples triggers the Windows Installer error.

Recommendations

For new-style InstallScript MSI installations, the UI functionality may be run without administrator privileges. Therefore, the following points are recommended (and, in general, also apply to all Windows Installer–based installations):

- Do not assume that administrator privileges will be available from any InstallScript events.
- Any system changes that need to be made should be done with custom actions in the Installation Execute sequence. InstallScript custom actions can be used. To ensure that sufficient privileges are available, custom actions should have an in-script execution setting of deferred in system context.
- Custom actions that modify the system should have corresponding rollback actions that restore the system to its earlier state if the installation fails.
- Custom actions that modify the system should also have corresponding actions that run during uninstallation to remove the installed items from the system.
- If changes are made to the system from InstallScript events, the changes are not logged by the InstallScript engine and would need additional script code to clean up these resources during uninstallation.

Using InstallScript to Differentiate Between the Two Styles at Run Time

You can use the INSTALLSCRIPTMSIEEUI variable to write a single script that produces different run-time behavior for the different UI styles. To learn more, see INSTALLSCRIPTMSIEEUI.

Specifying the InstallScript User Interface Type for InstallScript MSI Installations



Project • *This information applies to InstallScript MSI projects.*

In InstallScript MSI projects, you have the choice of two different types of InstallScript user interface (UI). For detailed information about these two styles, including any limitations, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.



Task ***To specify the InstallScript user interface type for an InstallScript MSI project:***

1. In the View List under **Installation Information**, click **General Information**.
2. For the **InstallScript User Interface Type** setting, select the appropriate option.
   - **Traditional Style (Requires Setup.exe)**—If you want to use the InstallScript engine as an external UI handler for your InstallScript MSI installation, select this option. With this style, your installation must include a **Setup.exe** setup launcher. The setup launcher serves as a bootstrap application that initiates the InstallScript engine to display the UI and run the InstallScript code, and the Windows Installer to run the Execute sequence of the .msi package.

     This is the default option for this setting.
   - **New Style (Requires Windows Installer 4.5)**—If you want to use the InstallScript engine as an embedded UI handler for your InstallScript MSI installation, select this option. With this style, InstallShield embeds the InstallScript engine within the .msi package. The Windows Installer calls the InstallScript engine to display the UI. The Windows Installer also runs the Execute sequence of the .msi package.

     This option requires Windows Installer 4.5 on the target machine. This option also has some limitations that require careful planning if you decide to use this style.

Entering Summary Information Stream Data



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Some of the Summary Information Stream settings are available in DIM projects. In this project type, the values in these settings are not displayed or used at run time. You can use these settings in this project type as a reference for internally identifying the project.

Windows Installer databases (.msi and .msm files) are implemented as COM structured storage, and COM structured storage files usually contain a Summary Information Stream. The Summary Information Stream contains information about your company and the software being installed. For more information, see Accessing the Summary Information Stream Panel to learn how to view your file's summary information.

For a description of each Summary Information Stream setting, see General Information View.

Accessing the Summary Information Stream Panel



Task ***To access summary information for an .msi or .msm file:***

1. Right-click the .msi file or .msm file and click **Properties**. The **Properties** dialog box opens.
2. Click the **Summary** tab to view the summary information.



Tip *• The General Information view is where you populate the information for the Properties dialog box for your installation.*

Setting Summary Information Stream Properties



Task ***To specify summary information for the project:***

1. In the View List under **Installation Information**, click **General Information**.
2. Find the **Summary Information Stream** area of the view.
3. Modify the values of the settings as needed.

Using the Template Summary Property



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You can use the Template Summary setting in the General Information view to essentially establish conditions for processor type and language; if a target system does not meet these requirements, the installer displays an error message and exits the installation.



Tip *• You can also set the Template Summary setting for a product configuration in the Releases view. Any value entered in the Releases view overrides the value set in the General Information view.*

Syntax

List the processor type first, followed by your installation's default language. For language, enter any four-digit language ID or 0 (zero) for language neutral.

- Separate the two categories with a semicolon.
- If your installation supports multiple products and you want to have multiple entries in the language category—for example, 1033 for English and 1031 for German—separate them with a comma.



Caution *• If your installation targets 64-bit machines, do not enter both Intel and Intel64 in this field. Doing so causes your installation to fail. Also, if you enter Intel64, your installation will not run on 32-bit operating systems.*

Valid processor values include:

- Intel
- Intel64
- x64

Setting Conditions Based on Processor Type and Language

If your installation runs only on Intel processors and English-based systems, use the following syntax in the Template Summary property:

Intel;1033

If the processor-type condition fails, the installer displays an error message and then exits.

If your product runs on x64 processors and supports English and German, enter the following:

x64;1033,1031

To specify that the package is language-independent, you can use the syntax **Intel;** or **Intel;0**.

Setting Conditions Based on Language Only

To put a condition on the language only, start the entry with a semicolon and specify your setup's default language:

;1036

Leaving the first portion empty indicates that the target processor is Intel. If a language is not specified, InstallShield provides the release languages in the Summary Information Stream.

Configuring Add or Remove Programs Information

The settings in the Add or Remove Programs area of the General Information view describe the information that appears in Add or Remove Programs tool in the Control Panel.

If you have not entered a value for a particular Add or Remove Programs setting, a sample value appears in grey text. This value is meant to serve as an example and is not included in the final Windows Installer database.

The General Information view enables you to set much of the information that is displayed to the end user through Add or Remove Programs.



***Project** • In Basic MSI and InstallScript MSI projects—Most of the values that you specify are stored in Windows Installer properties with names beginning with ARP, such as **ARPPRODUCTICON** or **ARPHELPLINK**. One exception to this is the Publisher setting. This value should be set with your company name and is stored in the **Manufacturer** property.*

*In a Basic MSI project—To prevent your product from appearing in the Add or Remove Programs, you can set the Windows Installer property **ARPSYSTEMCOMPONENT** to 1 in the Property Manager. Note that setting this property simply suppresses the display of your product in Add or Remove Programs. An end user can still remove your product by running the installation in maintenance mode or from the command line.*

In an InstallScript MSI project—To prevent your product from appearing in the Add or Remove Programs, select Yes for the Hide Add/Remove Panel Entry setting in the Releases view.

Specifying a Readme File



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Suite/Advanced UI*
- *Transform*

In the Read Me setting in the General Information view, enter the name of the Readme file for your product. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Add or Remove Programs. Alternatively, you can link to a Readme file located on the Internet by specifying a valid URL.



***Project** • In a Windows Installer–based project (Basic MSI or InstallScript MSI project), special formatting is required to properly display a path when the Readme file is installed as part of your installation; see the following section. In an InstallScript or InstallScript Object project, enter a hard-coded path or a URL, or specify a path relative to a system variable enclosed by angle brackets, for example, **<TARGETDIR>\Readme.txt**. This data is written to the target system's registry by the default **OnMoveData** event handler.*

Displaying the Readme File Path in Basic MSI and InstallScript MSI Projects

You can enter a hard-coded path or a URL for the Readme file. However, the name of a folder, such as **[MyDirectory]**, or of a component, such as **[$MyComponent]**, is not resolved. That means that the Readme value in the Support Information dialog box would be displayed as **[INSTALLDIR]Readme.txt** or **[$MyComp]Readme.htm**. The path is not resolved because this property is merely setting the initial value of the Windows Installer property ARPREADME, and formatted strings are never resolved for Windows Installer properties.

InstallShield's solution is to use a custom action to set **ARPREADME** with the value that you enter in the Read Me setting. By setting it with the SetARPReadme custom action, the path is resolved during the installation. For example, **Readme.doc** might be a file in the component Help_Files. In this case, enter **[$Help_Files]Readme.doc** for the Read Me setting. Assuming Help_Files is installed to **C:\Program Files\MyCompany\MyProduct\Help**, the path that would be displayed in Add or Remove Programs on the target system would be **C:\Program Files\MyCompany\MyProduct\Help\Readme.doc**.

Instead of the component name, you can use the file key. Do not use a folder from the **Directory** table because it will not be resolved at run time. The file key is not as reliable as the component name, because it is subject to change if the files are dynamically linked or the release uses patch optimization. Continuing the example above, if **Readme.doc** has a file key of F501_Readme.doc, enter **[#F501_Readme.doc]** for the Read Me setting. If it is installed to the same folder, Add or Remove Programs displays the following path on the target system:

C:\Program Files\MyCompany\MyProduct\Help\Readme.doc

The custom action SetARPReadme is added every time that the Read Me setting in the General Information view contains a string and the Installation Execute sequence contains the CostFinalize action. SetARPReadme is inserted into the Installation Execute and User Interface sequences directly after CostFinalize.

Including a Software Identification Tag for Your Product



Project • *This information applies to Basic MSI projects.*

ISO/IEC 19770-2 is an international standard for the creation of software identification tags. A software identification tag is a small XML-based file that contains descriptive information about the software, such as the product name, product edition, product version, and publisher. Software asset management tools collect the data in the tags to provide accurate application identification for software that is installed in an enterprise.

Software identification tagging is evolving as an industry standard, enabling independent software vendors to create smarter applications that give their customers better information for software asset management and license optimization initiatives. Including the identification tag in your product's installation makes it possible for your customers to use tools that can monitor their internal usage of your product, allowing them to understand, manage, and optimize the number of licenses of your product that they obtain from you.

Proper tag creation requires that you configure standard General Information settings such as Product Name and Product Version. It also requires that you configure a few identification-specific settings, which are also in the General Information view.

Task ***To include a software identification tag in your installation:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Software Identification Tag** area of the view, modify the values of the settings as needed.

   The **Use Software Identification Tag** setting lets you specify whether you want to include a tag in your installation. Select **Yes**, which is the default value, and then configure the other settings in the **Software Identification Tag** area as needed.

When you use tagging in your project, InstallShield adds the tag to two new components that it creates, and it associates the components with one of your project's features. The components are:

- ISO19770_LocalTag, which has a destination of **INSTALLDIR**
- ISO19770_SystemTag, which has a destination of **CommonAppDataFolder**

Use the Setup Design view if you want to associate these components with a different feature in your project. For more information, see Component-Feature Associations.

At build time, if the following conditions are true, InstallShield includes the software identification tag with the installation that it builds:

- Yes, the default value, is selected for the Use Software Identification Tag setting in the General Information view.
- The Unique ID, Tag Creator, and Tag Creator ID settings in the General Information view have values.

Note that if tagging is enabled but you have not entered values in one or more of the three aforementioned tag identification settings, InstallShield generates a build warning to inform you that the tag could not be included in your release. To resolve this warning, configure the settings in the Software Identification Tag area of the General Information view as needed.

If you configure your project to include a software identification tag and you also configure the release in the Releases view to use a .pfx file to digitally sign your release, InstallShield digitally signs the tag at build time. Note that the .NET Framework 3.5 must be installed on your build machine in order to sign a tag file.

Including Microsoft System Center Configuration Manager Application Model Data About Your Product

Project • *This information applies to Basic MSI projects.*

Microsoft System Center 2012 Configuration Manager is a solution that helps IT administrators manage and deploy applications to users throughout an enterprise. It takes a user-centric approach to application delivery, allowing IT administrators to define each application so that it can be deployed to users on a variety of devices in the most appropriate format, with the required application dependencies.

Accurate identification of deployment metadata is necessary for migrating applications into the Configuration Manager application model. InstallShield includes the ability to specify some of the application model metadata for an application through a software identification tag. When AdminStudio users import a package into the AdminStudio Application Catalog, AdminStudio Application Manager mines package elements for deployment data

such as detection methods, dependencies, requirements, as well as information in the software identification tag. AdminStudio makes this information available to users for review and tests before publishing to Microsoft System Center 2012 Configuration Manager.



Task ***To include application model data for System Center Configuration Manager in your installation:***

1. Ensure that you are including a software identification tag in your project. For more information, see Including a Software Identification Tag for Your Product.
2. In the View List under **Installation Information**, click **General Information**.
3. In the **Add SCCM App Model Data** setting, select **Yes**.
4. Configure the subsettings under the **Add SCCM App Model Data** setting as needed. For more information, see General Information View Settings.

When you include tagging and application model data in your project, InstallShield adds the tag with the application model data to two new components that it creates, and it associates the components with one of your project's features. The components are:

- ISO19770_LocalTag, which has a destination of **INSTALLDIR**
- ISO19770_SystemTag, which has a destination of **CommonAppDataFolder**

Use the Setup Design view if you want to associate these components with a different feature in your project. For more information, see Component-Feature Associations.

At build time, if the following conditions are true, InstallShield includes the software identification tag with the installation that it builds; it also includes the application model data in the tag:

- Yes, the default value, is selected for the Use Software Identification Tag setting in the General Information view.
- The Unique ID, Tag Creator, and Tag Creator ID settings in the General Information view have values.
- Yes is selected for the Add SCCM App Model Data setting in the General Information view.

If you configure your project to include a software identification tag and you also configure the release in the Releases view to use a .pfx file to digitally sign your release, InstallShield digitally signs the tag at build time. Note that the .NET Framework 2.0 or later must be installed on your build machine in order to sign a tag file.

Organizing Files for Your Installation

The primary task of an installation is to transfer files from your distribution media to your end user's hard drive. In an InstallShield installation, files are organized in a hierarchy: files are included in components; components are associated with features (and optionally subfeatures); and, in InstallScript and InstallScript MSI installations, features are associated with setup types.

At run time, your end user simply selects the setup type—or, if you allow it, the features and subfeatures—that he or she wishes to install. End users do not see components, which you use to group your files according to their type and purpose. For example, files that are installed to the same target folder can be placed in the same component, as can self-registering files.

A file often relies on functions in other files to perform a task. However, you may not be aware of all these other files—known as dependencies—when you include your application's files in your project. To help you identify dependencies, InstallShield offers three dependency scanners that automatically add these files to your project.

In addition to including individual files in your project, you can also include redistributables (InstallShield prerequisites, merge modules, and objects), which contain logic and files needed to install distinct pieces of functionality. For example, to include the Java Runtime Environment (JRE) files in your installation, you can add the InstallShield prerequisite for JRE to your installation project.

Designing Installations

There are two main perspectives in an installation project—that of the developer and that of the end user. In InstallShield, these perspectives are addressed in detail in the Components and Features views, respectively.

Components

Components represent the developer's view of the product. They are installation authoring tools that help the developer organize similar application data—such as files, registry entries, and shortcuts—into logical groups.

To enable the end user to choose which features to install, you should divide your application into components that correspond with the features of your application.

Using InstallShield, you can also publish your project's components.



Note • *The conditions that you specify for the Publish Components, Publish Features, or Publish Product actions are not validated during design time, so use proper syntax and check to ensure that the condition produces the expected outcome.*

Features

Features are the building blocks of an application from the end user's perspective. Each feature represents a specific piece of functionality for your product—such as the help files. End users should be able to install and uninstall discrete features of your product.

For example, an end user with limited hard drive space could elect not to install a product tutorial. If the user subsequently purchases another computer or frees resources on an existing one, the previously uninstalled product tutorial could then be installed.

You should separate your application into features that correspond to the components of your application.

Separating Applications into Components

Use the following guidelines to separate an application into components.

- Identify resources that function as a group; each group can be a component.
- Windows Installer requires that every file in a component be installed to the same directory. If you need to install a directory structure, each subdirectory must correspond to a separate component.
- The resources in a component should require the same conditions.
- To take full advantage of Windows Installer repair functionality, each .exe, .dll, and .ocx file should be placed in its own component, and each should be the component's key file. The Best Practices option of the Component Wizard can create components for you using this rule.
- Similarly, each .chm or .hlp file should be placed in its own component, along with any corresponding .chi or .cnt file.
- No resource should be included in more than one component, even across products. If a component's resources are required by more than one product, consider creating a merge module with the shared resources.
- When mapping a component to one or more features, ensure that all parts of a component map to the features.
- When testing your installation, you may want to divide your application into several components to thoroughly measure its performance.

Separating Applications into Features

Use the following guidelines to separate an application into features.

- Separate your application into independent parts, such as help, clip art, and program files. This gives the end user combination options when installing your application. For instance, if your application contains a large, graphic-intensive clip art file, you could make the clip art file a feature. This gives the end user the option of installing or not installing the file—a vital ability for users with limited available resources.
- In separating your application into features, ensure that end users can recombine the parts in several ways to fulfill specific needs. When doing this, consider the needs of all users, from system administrator to customer service representative to developer and everyone in between. By addressing all groups of users, you are promoting increased distribution and usage of your application.
- Each feature should have one function—such as help files—and should be clearly defined according to this functionality to facilitate recognition and comprehension. Features should possess an independent functionality, such that users can install and use the feature by itself, if they so choose.
- If one feature requires another, make the dependent feature a child of the other.
- To eliminate confusion, keep any information pertaining to the management of the system or application transparent to the user.

Using Path Variables



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

Traditionally, to link to source files in an installation project, you would need to create a reference to that file using a hard-coded path. For example, you might have a source file called Program.exe located at **C:\Work\Files** that you want to include in your installation. However, if you used hard-coded paths, you needed to enter the entire path every time that you want to associate a source file from that directory. If you moved the file to another directory, you needed to change the hard-coded path as it appears in your installation project. If your installation consisted of a small number of source files, this might not have been a problem. Unfortunately, some installations contain thousands of files that all would need to be remapped if you change the folder structure or migrate the project to a different machine.

With path variables, you can define commonly used paths in a central location so that you do not need to change every source file's path each time you move the project or change the directory structure.

All path variables can be viewed and modified in the Path Variables view. You can use path variables in almost any location in InstallShield where you link to source files, such as in the Dialog Editor, dynamic file links, and the release location. Instead of entering the path variables yourself, you can have InstallShield recommend them whenever you browse to a path. To have InstallShield automatically recommend path variables, select the Always display the Path Variable Recommendation dialog to me option on the Path Variables tab of the Options dialog box.



***Note** • Path variables are used during the development of your installation project. These paths do not apply to the target machines where the application is being installed. Rather, they are used to link to source files for your installation project. When the project is built, those links are evaluated and the files they point to will be built into the installation.*

Types of Path Variables



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

InstallShield includes support for several types of path variables:

- Predefined Path Variables
- Standard Path Variables
- Registry Path Variables
- Environment Path Variables

Predefined path variables define standard Windows directories; this type of variable cannot be renamed or modified. The standard, registry, and environment types of path variables are custom variables that you can create and define as needed.

Predefined Path Variables



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

Predefined path variables are variables that are set to certain standard Windows directories. These variables cannot be renamed or modified, but they can be used in your installation project to point to predefined directories. Following is a list of predefined path variables, their typical values, and the corresponding InstallScript path variables. (In an InstallScript project, InstallScript path variables are used in InstallShield, for example, as options in the list for the Destination property of a component. InstallScript path variables also correspond to system variables that can be used in the installation script.)

Table 4-1 • Predefined Path Variables

Predefined Path Variable	Value	InstallScript Path Variable
<CommonFilesFolder>	**C:\Program Files\Common Files**	<COMMONFILES>
<ISProductFolder>	**C:\Program Files\InstallShield\2016**	
<ISProjectDataFolder>	**<ISProjectFolder>\ProjectName**	<ISPROJECTDIR>
<ISProjectFolder>	**C:\InstallShield 2016 Projects**	

Table 4-1 • Predefined Path Variables (cont.)

Predefined Path Variable	Value	InstallScript Path Variable
<ISRedistPlatformDependentExpressFolder>	**C:\Program Files\InstallShield\2016\Redist\Language Independent\i386 Express**	
<ISRedistPlatformDependentFolder>	**C:\Program Files\InstallShield\2016\Redist\Language Independent\i386**	
<ProgramFilesFolder>	**C:\Program Files**	<PROGRAMFILES>
<SystemFolder>	**C:\Windows\System32**	<WINSYSDIR>
<VSSolutionFolder>	Varies. This variable references a higher-level base directory. This support enables you to have in your InstallShield projects static links to files in sibling projects that are within the Visual Studio solution folder. To learn more, see Using the VSSolutionFolder Path Variable with Visual Studio Solutions.	
<WindowsFolder>	**C:\Windows**	<WINDIR>

Using Predefined Path Variables



Project • *Windows Installer based projects only: The Path Variable Recommendation dialog box enables you to specify whether you want to use predefined path variables. This dialog box is launched if you enter a hard-coded path when inserting a source file into your installation and you have selected the Always display the Path Variable Recommendation dialog to me check box on the Path Variables tab of the Options dialog box. If you enter a path that is included in a predefined path variable, InstallShield suggests that you use the path variable rather than the hard-coded path.*

You can also use predefined path variables when defining the value for a standard path variable. You may want to define a standard path variable to a subfolder of a predefined variable. For example, **<ProgramFilesFolder>\InstallShield** could be used as the value for a standard path variable.

Predefined path variables are *smart* variables. This means they are set to the correct directory, even if your Windows directory is located on your D drive instead of your C drive. If you change the default project location on the File Locations tab of the Options dialog box, InstallShield updates this variable to reflect the new directory.

Standard Path Variables



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

Standard path variables are sometimes referred to as user-defined variables. These variable types are native to InstallShield. That is, they do not rely upon outside resources, as do registry or environment variables.

Using Standard Path Variables

Once you have created a standard path variable, you can use it every time you add a source file to your project. For example, you might create a variable called *<MyFiles>* and have it pointing to **C:\Work\Files**. If you add a source file to your project that contains that variable's path in its destination, it is recommended that you use the variable rather than hard-coding the path. This recommendation is displayed in the Path Variable Recommendation dialog box.

To have InstallShield automatically recommend path variables, select the Always display the Path Variable Recommendation dialog to me option on the Path Variables tab of the Options dialog box.



Tip *• To override the value of a user-defined path variable, an environment variable, or a registry value for a particular release at build time, use the Path Variable Overrides setting on the Build tab for that release. To learn more, see Overriding the Value of a Custom Path Variable for a Release.*

Registry Path Variables



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

Registry path variables enable you to define your own variables based on the default value of a specified registry key.

Once you have created a registry path variable, you can use it every time that you add a source file to your project. When you add a file to a component from a folder that contains the value of *MyRegVar*, it is recommended that you use the variable rather than hard-coding the path. This recommendation is displayed in the Path Variable Recommendation dialog box.

To have InstallShield automatically recommend path variables, select the Always display the Path Variable Recommendation dialog to me option on the Path Variables tab of the Options dialog box.



Tip *• To override the value of a user-defined path variable, an environment variable, or a registry value for a particular release at build time, use the Path Variable Overrides setting on the Build tab for that release. To learn more, see Overriding the Value of a Custom Path Variable for a Release.*

Environment Path Variables



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

Environment path variables enable you to define your own path variables based on certain values on the Environment dialog box.



Task ***To view the Environment dialog box:***

1. Right-click **My Computer** and click **Properties**. The **System Properties** dialog box opens.
2. On the **Advanced** tab, click **Environment Variables**.

A common scenario where environment path variables may be useful is when performing a build from the command line. If you do nightly builds on a machine dedicated to that purpose, you may need to change the path variables from the command line. As long as you use environment path variables, you can define the paths to your project's files from either a batch file or the command line. These paths are evaluated when the installation project is built, and the correct paths to the files will be used.

Using Environment Path Variables

After you have created an environment path variable, you can use it every time that you add a source file to your project. For example, you might create a variable called *<MyFiles>* and have it point to **C:\Work\Files**. If you add a source file to your project that contains that variable's path in its destination, it is recommended that you use the variable rather than hard-coding the path. This recommendation is displayed in the Path Variable Recommendation dialog box.

To have InstallShield automatically recommend path variables, select the **Always display the Path Variable Recommendation dialog to me** option on the Path Variables tab of the Options dialog box.

Tip • *To override the value of a user-defined path variable, an environment variable, or a registry value for a particular release at build time, use the Path Variable Overrides setting on the Build tab for that release. To learn more, see Overriding the Value of a Custom Path Variable for a Release.*

Creating and Defining a Custom Path Variable

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

The Path Variables view is where you create and define standard path variables, environment path variables, and registry path variables.

Task ***To create and define a path variable:***

1. Do one of the following:
   - In QuickPatch projects: In the View List under **Patch Settings**, click **Path Variables**.
   - In all other project types: In the View List under **Media**, click **Path Variables**.
2. Do one of the following:
   - To create a standard path variable, click the **New Path Variable** button.
   - To create an environment variable, click the arrow next to the **New Path Variable** button, and then click **Environment**.
   - To create a registry variable, click the arrow next to the **New Path Variable** button, and then click **Registry**.

InstallShield adds a new row at the bottom of the view.

3. Complete each of the settings in the new row as needed.

For descriptions of each column, see Path Variables View.



Tip • *To override the value of a user-defined path variable, an environment variable, or a registry value for a particular release at build time, use the Path Variable Overrides setting on the Build tab for that release. To learn more, see Build Tab for a Release.*

Overriding the Value of a Custom Path Variable for a Release



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

InstallShield lets you override the values of your project's standard path variables, environment path variables, and registry path variables for each release in your project. This functionality enables you to essentially replace certain files and folders in your project with others at build time, depending on the particular release that you are building.

For example, in a Basic MSI project, you could use this functionality to swap out the custom action DLL files—a 32-bit version for your project's 32-bit release, and a 64-bit version for your project's 64-bit release. Or perhaps for a Suite/Advanced UI project, you might want to use this functionality to swap out UI elements such as images and EULAs for different releases in your project; this would enable you to easily generate installations for different editions or branded versions of your product.



Task ***To override the value of a custom path variable for a release:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the release that you want to configure.
3. On the **Build** tab, in the **Path Variable Overrides** setting, click the ellipsis setting (**...**). The **Path Variable Overrides** dialog box opens.

   The Path Variable Overrides dialog box shows check boxes for the path variables, environment path variables, and registry path variables that are configured in the Path Variables view. Note that it does not include check boxes for any predefined path variables, or for path variables that are already overridden for the same release in the Releases view.
4. Select the check box of each path variable that you would like to override for the selected release.

5. Click **OK**. InstallShield closes the dialog box. Under the **Path Variable Overrides** setting, InstallShield adds a new subsetting for each path variable that you selected.
6. For each path variable subsetting, specify the appropriate new value.

At build time, InstallShield uses the new values that you specified in the Releases view to override the default values that are set in the Path Variables view.

Additional Methods for Overriding Path Variables

As an alternative to overriding a path variable in the Releases view, you can override a path variable in the following ways:

- If you are building from the command line with `IsCmdBld.exe`, use the `-l` parameter to specify a path variable and a new value.
- If you are building through MSBuild or Team Foundation Server (TFS), use the PathVariables parameter on the InstallShield task. This parameter is exposed as the ItemGroup InstallShieldPathVariableOverrides when the default targets file is used.

Note those methods take precedence over any value that you set in the Path Variable Overrides setting in the Releases view.

Converting Absolute Path References in Your Project to Path Variables



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

You might not need to use path variables when you begin your installation project, but you may realize the benefits they provide well after you have started. Rather than going through each file link and changing it to a path variable, you can use the Converting Source Paths Wizard.

This wizard scans your project for static links and converts them all to path variables. If any of your static links can be replaced with existing path variables, the wizard does this automatically. For all other links, the wizard provides a list of the possible variables. After you choose the variables that you want to use, the wizard converts all of the links.



Task ***To launch the Convert Source Paths Wizard:***

On the **Project** menu, click **Convert Source Paths**.

Deleting a Path Variable



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*



Task ***To remove a path variable from your project:***

1. Do one of the following:
   - In QuickPatch projects: In the View List under **Patch Settings**, click **Path Variables**.
   - In all other project types: In the View List under **Media**, click **Path Variables**.
2. Select one or more path variables that you want to delete.

   To select multiple consecutive path variables, select the first path variable, press and hold SHIFT, and select the last path variable. To select multiple nonconsecutive path variables, select the first path variable, press and hold CTRL, and select each additional path variable.
3. Click the **Delete Selected Path Variables** button.

Including Files and Folders

Your files, the core of your product, are also the core of your installation. Files are added to a project only at the component level. In installation projects, components are associated with features, which are what end users see when they run your installation. If the component's feature is selected for installation, the component's files are installed to the target system.

When you are adding files to components in Windows Installer–based projects, you should be aware of Setup Best Practices. By default, the Setup Best Practices Wizard monitors your setup design and alerts you if you have violated Best Practices.

You can add files to your project in the Setup Design view (for installation projects) or the Components view, and in the Files and Folders view.

Adding to Your Project Files to Be Installed on Target Systems

You can use the Files and Folders view to add files to your project.



Task ***To add a file to your project:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **View Filter** list, select the feature that should contain the file that you want to add.
3. Windows Installer–based projects only: In the **Destination computer's folders** pane, right-click **Destination Computer**, point to **Show Predefined Folder**, and then click the predefined folder that you want to use.
4. In the **Destination computer's folders** pane, click the folder into which you want to place the file.
5. In the **Source computer's folders** pane, navigate to the folder containing the file you want to add.

   INSTALLDIR (for Windows Installer–based projects) or Application Target Folder (for InstallScript projects) is the most commonly used target location, because this is the default root directory for your application's files.
6. Select and drag the file you want to add from the **Source computer's files** pane to the **Destination computer's files** pane.



Tip • *You can specify how* ***INSTALLDIR*** *is displayed in the Destination computer's folders pane of the Files and Folders view by setting your preference on the Directory tab of the Options dialog box.*

You can also specify a hard-coded destination directory.

The Destination computer's folders pane contains a list of predefined destinations. You can add a file to either a predefined destination or to a destination that you create.

Automatically Creating New Components When You Add Files

InstallShield can create new components to properly handle the new files as you add them to your project. To accommodate this, use the View Filter in this view to select the feature with which you want to associate any newly created components.

If no features exist, you have the option of creating one when you first add files in this view.

Project-Specific Differences for Automatic New Component Creation

In a Windows Installer–based project, InstallShield creates components according to Setup Best Practices. For example, all portable executable files (.exe, .dll, .ocx files) are given their own component. All other files are added to the default component for each destination directory. However, if you drag files directly to one of the components listed under a destination folder, the Setup Best Practices rules are ignored.

In an InstallScript project, InstallShield creates components based on the file types—for example, all self-registering files are contained in one component, all non-self-registering files in another component, and all .NET files in a different component. The components appear as subfolders in the Files and Folders view.

Changing the Component/Feature Relationship

You can change the component/feature relationship by right-clicking a component in the **Destination computer's folders** pane and then clicking Properties. On the Properties dialog box, use the Features tab to make the necessary changes.

Tip *• To display components in the Files and Folders view, right-click a destination folder (in the lower-left pane of the view) and click Show Components.*

Dragging and Dropping Files Using the Context Menu

The files explorer in the Files and Folders view enables you to drag and drop folders from your source computer to destinations on the target system. You have a number of options when you drag files from the **Source computer's folders** pane to the **Destination computer's folders** pane.

Tip *• When you are using InstallShield on a 64-bit system and you want to add to your project a system file whose source location is the 64-bit System folder (System32) on your development machine, it is not possible to drag the file from one of the source computer panes at the top of the Files and Folders view to the appropriate location in one of the destination computer panes. To learn more, see Adding Files from Your 64-Bit Source Machine's 64-Bit System32 Folder.*

Task ***To display the context menu:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **Source computer's folders** pane or the **Source computer's files** pane, right-click a folder or file and drag it to the **Destination computer's folders** pane or the **Destination computer's files** pane. Then release the mouse button.

   If the predefined folder to which you adding the folder or file is not displayed in the **Destination computer's folders** pane, you can add it. To learn how, see Displaying Predefined Folders in the Files View.

InstallShield displays a context menu that contains several commands.

Table 4-2 • Commands that Are Available on the Context Menu in the Files and Folders View

Command	Description
Add	Adds the folder, subfolders, and/or files selected. This is the same as the default drag-and-drop behavior.
Add Preserving Source Structure	Adds the selected folder, subfolders, and/or files while preserving the file/folder structure found on the source computer. This command is available only if the source folder matches a predefined Windows destination folder. In addition, the destination on which you drop the file or folder must be the Destination Computer.
Add Folder(s) Only	Adds only the folders selected and any subfolders contained in the selected folder. Does not add any files contained within the selected folders or subfolders.

Table 4-2 • Commands that Are Available on the Context Menu in the Files and Folders View (cont.)

Command	Description
Add Folder(s) Only Preserving Source Structure	Adds only the folders selected and any subfolders contained in the selected folder. Does not add any files contained within the selected folders or subfolders. This option also preserves the folder structure found on the source computer. This option is available only if the source folder matches a predefined Windows destination folder. In addition, the destination on which you drop the file or folder must be the Destination Computer.
Cancel	Ends the drag-and-drop operation without making any changes.

Adding Files from Your 64-Bit Source Machine's 64-Bit System32 Folder

On 64-bit systems, the System32 folder is reserved for 64-bit applications. When you try to view your development machine's 64-bit System folder from within the Files and Folders view in InstallShield, Windows redirects the view to instead display the SysWOW64 folder—the 32-bit version of the folder. Thus, when you are using InstallShield on a 64-bit system and you want to add to your project a system file whose source location is the 64-bit System folder (System32) on your development machine, it is not possible to drag the file from one of the source computer panes at the top of the Files and Folders view to the appropriate location in one of the destination computer panes.

To work around the redirection and add a 64-bit System32 file on your development machine to your InstallShield project, you can browse to the Sysnative folder on your machine and then select the appropriate file for your project. The following instructions explain how.

Caution • *Note that in many cases, including system files in an installation is not recommended, since the system folder is protected by Windows. In these cases, the preferred way to deliver and update system files is to use a Microsoft redistributable, if one is available for the technology, or to have end users obtain the updates through Windows Update.*

Task ***To add to your project a file in the 64-bit System32 folder of a development system that has 64-bit Windows:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, click the folder into which you want to place the file.
3. Right-click in the **Destination computer's folders** pane, and then click **Add File**. The **Open** dialog box opens.
4. Specify the following path (but replace the drive letter with the applicable drive letter if appropriate):

   C:\Windows\Sysnative
5. Select the appropriate file that you want to add to your project, and then click the **Open** button.

InstallShield adds the file to your project. InstallShield uses the Sysnative folder as part of the path for the source file that you added. WOW64 sees the Sysnative folder as a special alias; the file system does not redirect access away from this folder.



Tip *• Note that you can also add 64-bit System32 files through the Components view or the Setup Design view: in these views, expand the node of the component that you want to contain the 64-bit System32 file. Select the Files node under that component. Then, right-click the Files pane on the right, and then click Add File. Specify the C:\Windows\Sysnative path, as described in the aforementioned procedure.*

Availability of Sysnative Folder Support on Windows-Based Systems

Use of the Sysnative folder is not supported on 32-bit machines. If you use the Sysnative folder in an InstallShield project on a 64-bit system but then you try to build a release in that InstallShield project on a 32-bit system, InstallShield generates one or more build errors or warnings informing you that it could not find the source file.

If you are using the automation interface with InstallShield or the Standalone Build to create, edit, or build an installation on a 64-bit machine that has Sysnative support, you can use the Sysnative folder in paths when you are specifying the source folder for system files that are in the 64-bit System folder.

Creating New Destination Folders



Task ***To create subfolders of the predefined folders:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, right-click a folder and click **New Folder**.
3. Rename your new folder by selecting it, pressing F2, and typing the new name.

Alternately, you can drag an entire folder onto one of the destination folders, thereby adding a new subfolder to the target folder that contains the source folders files. This new subfolder has the same name as the source folder.

Creating Empty Folders

Basic MSI Projects

You can use the CreateFolder table of an .msi database—exposed in the Direct Editor—to create empty directories.

For an example, see Knowledge Base article Q103218.

InstallScript MSI Projects

The **CreateDir** function creates an empty directory.

Specifying Hard-Coded Destination Directories

In the Files and Folders view, you can specify hard-coded destination directories.

Specifying Drive Destinations



Task ***To specify a particular drive:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, right-click **Destination Computer** and click **New Folder**, or click **Destination Computer** and press INSERT. InstallShield adds a new folder.
3. For the name of the folder, type the drive letter followed by a colon (for example, **C:**).
4. Press ENTER.

Specifying Folders and Subfolders to Create a Destination Path

You can specify folders and subfolders beneath a drive letter folder to create a hard-coded destination path:

1. Right-click the drive folder (for example, **C:**) under which you want to add a folder or—to add a subfolder—right-click the folder under which you want to add a subfolder. You can also click the folder and press INSERT.
2. For the name of the folder, type the folder name, and then press ENTER.

Quickly Creating Destination Paths



Task ***To quickly create a nested destination path:***

1. Right-click a folder and click **New Folder**, or click a folder and press INSERT.
2. For the name of the folder, type the destination path—for example, **a\b\c**. This creates a folder structure with **a** at the top, **b** below **a**, and **c** below **b**.

Removing Files and Folders from Target Systems



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield has built-in support that makes it easy to specify files and folders that you want to be removed from target systems at run time. This file and folder removal capability is useful for scenarios such as removing application-created files that your installation does not otherwise track.

You can schedule the removal of a file or folder for one of the following events:

- When the file or folder's component is being installed
- When the file or folder's component is being uninstalled
- When the file or folder's component is being installed or uninstalled

Note that if the item to be removed is a folder, that folder is removed only if it is empty.

InstallShield offers different methods for configuring a file or folder removal in a project.

Using the Files and Folders View to Specify Files and Folders to Be Removed



Task ***To configure file and folder removals through the Files and Folders view:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, select the destination folder that contains the file or folder that you want to be removed.
3. Right-click in the **Destination computer's files** pane and click **Add File Removal**. The **Properties** dialog box opens.
4. Configure the settings as needed. For details, see File Removal Properties Dialog Box.

InstallShield adds a file or folder icon to the **Destination computer's files** pane. The icon has a red X through it to indicate that it is referencing an item to be removed.

Using the Setup Design View or Components View to Specify Files and Folders to Be Removed



Task ***To configure file and folder removals through the Setup Design view or the Components view:***

1. In the View list under **Organization**, click **Setup Design** (in installation projects) or **Components**.
2. In the **Setup Design** or **Components** tree, expand the node of the component that contains the file or folder that you want to be removed, and then click the **Files** subnode.
3. Right-click in the **Files** pane and click **Add File Removal**. The **Properties** dialog box opens.
4. Configure the settings as needed. For details, see File Removal Properties Dialog Box.

InstallShield adds a file or folder icon to the Files pane. The icon has a red X through it to indicate that it is referencing an item to be removed.

Editing File or Folder Removal Settings

To change the settings for a file or folder removal, right-click the removal item and then click Properties. The Properties dialog box opens, enabling you to edit the settings as needed.

Tips for Managing Files and Folders in Your Project

Through drag-and-drop operations, common keyboard shortcuts such as CTRL+C and CTRL+P, and context menus that are available when you right-click items, you can easily move the files and folders to different destination folders and to different features.

Managing the Destination of Files and Folders in Your Project



Task ***To manage the destination of files and folders in your project:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. Do any of the following:
   - To change the settings for a file or folder, right-click the item and then click **Properties**. The **Properties** dialog box opens, enabling you to edit the settings as needed.
   - To move a file from one destination folder to another, drag it from one location in the **Destination computer's files** pane to the appropriate folder in the **Destination computer's folders** pane.
   - To move a folder from one destination folder to another, drag it from one location in the **Destination computer's folders** pane to the appropriate folder in the same pane.
   - To copy a file or folder from one location to another, press and hold CTRL while dragging the item from one location to another.

     As an alternative, you can copy an item to your Clipboard and then paste it into the appropriate location. Right-click the item and click **Copy**, or click it and press CTRL+C. Next, select another folder that you want to contain the file. Then right-click its folder and click **Paste**, or click the folder and press CTRL+P.
   - To delete a file or folder from your project, right-click it and then click **Delete**.

Managing the Feature Associations for Files in Your Project



Task ***To manage the feature associations of files in your project:***

1. In the View list under **Application Data**, click **Setup Design** (in installation projects) or **Components**.
2. Expand the node of the component that contains the file whose feature association you want to manage, and then click the **Files** subnode. InstallShield displays its files in the **Files** pane on the right.
3. Do any of the following:
   - To copy a file from one feature to another, you can copy a file to your Clipboard and then paste it into the appropriate feature. Right-click the file and click **Copy**, or click it and press CTRL+C. Next, select another feature that you want to contain the file. Then right-click its Files pane, and click **Paste**, or click it and press CTRL+P.
   - To remove a file from a feature, right-click the file and then click **Remove**.

Finding Files and Folders in Your Project

If you have added numerous folders and files to your project, you may have trouble finding a particular folder or file. You can perform a search for folders and files in the Files view; InstallShield locates any matches and highlights the first one. You can keep searching until you find all matches for your search criteria.

Task ***To find files and folders in your project:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, select **Destination Computer**.
3. On the **Edit** menu, click **Find**. The **Find** dialog box opens.

   As an alternative, you can press CTRL+F.
4. In the **Find What** box, type the text to be found. You can use wildcard expressions, such as ***.exe**.
5. In the **Look at** area, specify whether you want to search for files, folders, or both.
6. Specify any other desired criteria.
7. Click **Find Next**. The first item (if any) that matches your search criteria is selected in either the **Destination computer's folders** pane or the **Destination computer's files** pane.
8. To find the next item (if any) that matches your criteria, press the F3 key. Repeat this step as necessary.

Displaying Components in the Files and Folders View

Task ***To display the components in your installation and the files that are associated with them:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, right-click any folder and click **Show Components** (in Windows Installer–based projects) or **Show Components and Subfolders** (in InstallScript-based projects).

Components in installation projects that are not associated with a feature are displayed with the orphaned component icon ( ).

Modifying Components through the Files and Folders View

In the folders explorer of the Files and Folders view, you can right-click a component and click Properties to display the Component Properties dialog box. Through this dialog box, you can modify the component's properties, add dynamic file links, and change that features contain the component.

Note *• (Windows Installer–based projects only) You can launch the Component Wizard by right-clicking a folder icon and then clicking Launch Component Wizard.*

Configuring Settings for a File

You can set a number of properties for each file that is to be installed on the target system.

Task ***To specify the properties for a file:***

1. In the View list under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, click the folder that contains the file whose properties you want to configure.
3. In the **Destination computer's files** pane, right-click the file and then click **Properties**. The **Properties** dialog box opens.

To see a description of each property that you can set, see File Properties Dialog Box.

File Associations

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

File associations are registry settings that tell Windows what application to use to open files of a certain type. For example, Windows typically launches Notepad.exe when a text (.txt) file is opened.

To view and modify registered file types on your system, open Windows Explorer, and on the Tools menu, click Folder Options. Use the File Types tab of the Folder Options dialog box to see how the file associations are configured.

Similarly, you can identify the application that is associated with a given file by right-clicking the file in Windows Explorer and then clicking Properties.

File associations are stored in both HKLM\SOFTWARE\Classes and HKCU\SOFTWARE\Classes; you can see a merged view of the data under HKEY_CLASSES_ROOT.

Creating File Associations for Your Installation Project

Best-practice guidelines recommend that you create a file association for every nonhidden type of file created or used by your product. You can quickly and easily create file associations for your installation project with the File Extensions view, which is available as an advanced setting within the Components view and the Setup Design view (for installation projects only). When an end user installs a feature that contains a file association, the file association is registered on the target machine; an entry is made in the appropriate part of the registry, and the entry links your

file type to your application through the ProgID. The ProgID, which is sometimes called a file type's application identifier or tag name, uniquely identifies your application and helps the operating system recognize your file association.

Adding New File Extensions



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If your application manipulates files with a unique file extension, you can register your file type. For example, if your application manipulates files with the .xyz extension, registering the file type instructs the operating system to open the file with your application when the user double-clicks its icon.



Task ***To add a new file extension:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.
4. Click **File Types**. The **File Types** explorer opens.
5. In the **File Types** explorer, right-click **Extensions** and click **New Extension**. InstallShield creates a new extension with the default name **New ExtensionN**, where *N* is a unique number.
6. Type your extension without the dot (for example, enter **ext** instead of **.ext**).

Removing File Extensions



***Project** • This information does not apply to InstallScript or InstallScript Object projects.*



Task ***To remove an extension:***

1. Open the **Setup Design** view (for installation projects only) or the **Components** view.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.

4. Click **File Types**. The **File Types** explorer opens.
5. In the **File Types** explorer, right-click the extension and click **Delete**.

Setting File Extension Properties



Project *• This information does not apply to InstallScript or InstallScript Object projects.*

The extension properties determine the registry entries for your file extension, as well as the ProgID associated with it. Detailed help is available in the help pane in InstallShield when you click each property.

Specifying Command Verbs



Project *• This information does not apply to InstallScript or InstallScript Object projects.*

By default, the verb open is added to your file extension.



Task ***To add a new verb and specify its properties:***

1. Open the **Setup Design** view (for installation projects only) or the **Components** view.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.
4. Click **File Types**. The **File Types** explorer opens.
5. In the **File Types** explorer, right-click the extension and click **New Verb**. InstallShield creates a new verb with the default name **New VerbN**, where *N* is a unique number.
6. Type the name of your new verb. You can use any of the canonical verbs, such as **open**, **print**, **find**, and **properties**.
7. Select the new verb to edit its properties. Its property sheet opens to the right.

Removing a Verb from a File Extension



Project *• This information does not apply to InstallScript or InstallScript Object projects.*



Task ***To remove a verb:***

1. Open the **Setup Design** view (for installation projects only) or the **Components** view.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.

4. Click **File Types**. The **File Types** explorer opens.
5. Right-click the verb and click **Delete**.

Adding New MIME Types



Project • *This information does not apply to InstallScript or InstallScript Object projects.*



Task ***To add a new MIME type:***

1. Open the **Setup Design** view (for installation projects only) or the **Components** view.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.
4. Click **File Types**. The **File Types** explorer opens.
5. In the **File Types** explorer, right-click the extension and click **New MIME Type**. InstallShield creates a new MIME type with the default name **MIME TypeN**, where *N* is a unique number.
6. Type the MIME type; for example, **image/jpeg**.

Click the MIME type to view its Class ID property. Click the Class ID property to edit it in the lower pane. Press CTRL+V to paste a class ID into the property field.

Removing MIME Types



Project • *This information does not apply to InstallScript or InstallScript Object projects.*



Task ***To remove a MIME type:***

1. Open the **Setup Design** view (for installation projects only) or the **Components** view.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.
4. Click **File Types**. The **File Types** explorer opens.
5. Right-click the MIME type and click **Delete**.

Creating ProgIDs



Project • *This information does not apply to InstallScript or InstallScript Object projects.*

The ProgIDs item in the File Types explorer contains all of the programmatic identifiers—ProgIDs, also known as application identifiers or tag names when they support file types—that you want to associate with file extensions. A file type's ProgID is an arbitrary string, but it should be unique on the target system. One ProgID naming convention is to append the word **file** to your extension without a dot—the .ext extension might use the ProgID *extfile*. Another convention is to name a file-type ProgID after the application used to open the file type, as in SampleApp.Document.

For example, an *.xyz* file extension could point to a xyzfile ProgID, and all of the *.xyz* file-type information would be registered under xyzfile.

Task ***To create a ProgID:***

1. Open the **Setup Design** view (for installation projects only) or the **Components** view.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.
4. Click **File Types**. The **File Types** explorer opens.
5. In the **File Types** explorer, click the extension. The extension properties are displayed in the right pane.
6. In the **ProgID** property, type the name of your ProgID. InstallShield adds your ProgID under **ProgIDs** in the **File Types** explorer.
7. In the **File Types** explorer, click the new ProgID and then set its properties.

Removing ProgIDs

Project • *This information does not apply to InstallScript or InstallScript Object projects.*

Task ***To remove a ProgID:***

1. Open the **Setup Design** view (for installation projects only) or the **Components** view.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.
4. Click **File Types**. The **File Types** explorer opens.
5. In the **File Types** explorer, click the extension. The extension properties are displayed in the right pane.
6. In the **ProgID** property, delete the ProgID.

Dynamic File Linking

If you want to add the contents of an entire directory to your project, you can do so through the use of dynamic file linking. When you select a source folder for dynamic linking, InstallShield adds the files within that folder to your release at build time. InstallShield scans the source folder before every build and automatically incorporates any new or changed files into your release. Dynamic file linking is useful when the list of files in a folder—and possibly the list of files in its subfolders—might change between builds.



Important • *Dynamic file linking should be used with caution. If you inadvertently delete a dynamically linked file from the source folder that your dynamic link references, that file is not included in your release the next time you build it, and InstallShield does not display any build warning or error. Your product may install without any issues, but it may not work as expected, since the dynamically linked file that was inadvertently deleted is no longer being installed. Therefore, it is recommended that you avoid using dynamic file links for critical executable files—such as .exe, .dll, or .ocx files—especially if your product requires them in order to run successfully.*

Whenever possible, it is better to use the best practice method, instead of the by-directory method, for creating components for dynamically linked files. Note that with both methods, however, a minor upgrade, a small update, or a patch may not install correctly if a file that was present in a target image is removed from the dynamic link for the upgrade or patch.

Filtering Dynamically Linked Files

When you are configuring your dynamic file link, you can specify whether you want to include subfolders of the dynamically linked folder. To further filter the files that are dynamically linked, you can identify specific names of files that you want to be included in or excluded from the dynamic link. In addition, you can use wild cards to specify only certain files or file types to be added or excluded.

For example, if all of your image files are in one folder along with sound files and you want to dynamically link only the image files, you could specify that you would like to include only .bmp and .ico files in the dynamically linked folder. To do so, you would use an asterisk (*) in your include pattern, as in the following example:

***.bmp, *.ico**

To include or exclude a specific file, you would enter the full file name in the include or exclude pattern box. For more details, see Creating a Dynamic Link.

Distinguishing Dynamically Linked Files and Folders from Static Files and Folders in the InstallShield Interface

When a dynamic file is displayed within the InstallShield interface, the lower-left corner of the file's icon includes an image that indicates that it is a dynamically linked file:



InstallShield includes that same dynamic file image on the icon of subfolders that are included in dynamic file links:



Furthermore, InstallShield adds an arrow to the folder image—in addition to the dynamic file image—on the component icon of subfolders that are included in dynamic file links:



The icons that the InstallShield interface displays for static files and folders do not include this dynamic link image.

Limitations of Dynamic File Linking



Important *• Dynamic file linking should be used with caution. If you inadvertently delete a dynamically linked file from the source folder that your dynamic link references, that file is not included in your release the next time you build it, and InstallShield does not display any build warning or error. Your product may install without any issues, but it may not work as expected, since the dynamically linked file that was inadvertently deleted is no longer being installed. Therefore, it is recommended that you avoid using dynamic file links for critical executable files—such as .exe, .dll, or .ocx files—especially if your product requires them in order to run successfully.*

Whenever possible, it is better to use the best practice method, instead of the by-directory method, for creating components for dynamically linked files. Note that with both methods, however, a minor upgrade, a small update, or a patch may not install correctly if a file that was present in a target image is removed from the dynamic link for the upgrade or patch.

Limitations for Windows Installer–Based Projects

Note the following limitations if you are considering dynamic file linking in Basic MSI, DIM, InstallScript MSI, and Merge Module projects:

- You cannot create custom actions to a dynamically linked file.
- You cannot associate a file extension with a dynamically linked file.
- You cannot extract COM information from a dynamically linked file.
- You cannot set any properties such as Shared, Permanent, or Never Overwrite for a dynamically linked file.
- You cannot use the .NET installer class functionality for a dynamically linked file.
- You cannot specify that COM interop should be enabled for a dynamically linked file.
- You cannot change default file settings (such as Read-Only or Hidden).
- You cannot set file permissions for a dynamically linked file.
- You cannot create shortcuts to a dynamically linked file.
- You cannot perform a static or dynamic scan of a dynamically linked file.
- For Basic MSI projects: You cannot launch a dynamically linked file from the SetupCompleteSuccess end-user dialog.

Any file that you add directly (not through dynamic linking) to your project has an internal name (FileKey). When you create a custom action, a file extension, shortcut, or other type of item, it actually points to this internal name.

When you add files to your project through dynamic links, the files are not physically added to the project. This means that these files do not have any FileKeys that can be associated with custom actions, file extensions, etc.

Limitations for InstallScript–Based Projects

For InstallScript and InstallScript Object projects, a single component cannot contain both static files and dynamically linked files.

Determining the Appropriate Component Creation Method for Dynamically Linked Files



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

InstallShield provides two methods for creating components for dynamically linked files: the best practice method and the one-component-per-directory method.

Using the Best Practice Method

When best practices for component creation are followed, InstallShield performs the following tasks at build time for all of the files that meet the include and exclude filter criteria of your dynamic link:

- InstallShield creates a separate component for each portable executable (PE) file in the dynamically linked folder. Each PE file is the key file of its component.
- InstallShield adds all non-PE files at the root level of the dynamic link to the component that contains the link.
- If the dynamic link includes a subfolder, InstallShield creates a new component for all of the non-PE files in that subfolder. If the dynamic link includes more than one subfolder, InstallShield creates a separate component for all of the non-PE files in each subfolder.

This is the default functionality for all new dynamic links.



Tip *• The File Extensions tab on the Options dialog box is where you specify which file types are PE files.*

Using the By-Directory Method

When the by-directory method is used for component creation, InstallShield performs the following tasks at build time for all of the files that meet the include and exclude filter criteria of your dynamic link:

- InstallShield creates one component for all of the files that are in the root-level dynamically linked folder, regardless of the file types.
- If the dynamic link includes one or more subfolders, InstallShield creates a separate component for all of the files in each subfolder, regardless of the file types. The first dynamically linked file in a subfolder's component is the key file of that component.

This method of component creation is the traditional method that was available in InstallShield before the best practice method was introduced.

Determining Which Component Creation Method to Use

For most dynamic links, the preferred component creation method is the best practice method. This method enables you to create patches more easily than with the by-directory method. For minor upgrades and small updates, the components, key files, and feature-component organization need to be maintained across the earlier and later .msi databases; for patches, the **File** table keys also need to be maintained. Since each component name and component

code—and possibly key files—change at each build with the by-directory method of dynamic file linking, issues may occur. The advantage of the best practice method is that it allows for greater predictability than with the by-directory method.



***Tip** • When you are configuring an upgrade, use the patch optimization functionality in your build settings. Using patch optimization helps you keep component names, component codes, **File** table keys, and **Directory** table keys synchronized from the earlier release. For more information, see Upgrade Considerations.*

Note that if you want to create a patch for an earlier version of your product, and the earlier installation includes dynamic links that used the by-directory method, you must continue to use the by-directory method for the same dynamic links. However, if you add new dynamic links in your upgrade project, you can use the best practice method for those new dynamic links. That is, you can mix both types of dynamic linking in the same project and create a patch to deliver the upgrade.



***Important** • Whenever possible, it is better to use the best practice method, instead of the by-directory method, for creating components for dynamically linked files. Note that with both methods, however, a minor upgrade, a small update, or a patch may not install correctly if a file that was present in a target image is removed from the dynamic link for the upgrade or patch.*



***Note** • For information on the rules that Windows Installer uses when determining whether a file included in a package should overwrite a file that already exists on the target system, see Overwriting Files and Components on the Target System.*

Specifying Which Component Creation Method You Want to Use

The File Linking tab on the Component Properties dialog box is where you specify which component creation method you want to use.

Creating a Dynamic Link



***Project** • The procedure for creating dynamic file links differs, depending on which project type you are using. Note that parts of the procedure apply to the following Windows Installer–based projects:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Parts of the procedure apply to the following InstallScript-based projects:

- *InstallScript*
- *InstallScript Object*

You can use the Files and Folders view to create a dynamic link.

Task ***To create a dynamic link:***

1. In the View List under **Application Data**, click **Files and Folders**.
2. In the **View Filter** list, select the feature that should contain components with the dynamically linked files.
3. Ensure that components are displayed in the **Destination computer's folders** pane.

   If components are not displayed: In the **Destination computer's folders** pane, right-click **Destination Computer** and click **Show Components** (in Windows Installer–based projects) or **Show Components and Subfolders** (in InstallScript-based projects).
4. For a Windows Installer–based project: right-click a component and click **Dynamic File Linking**. The **Component Properties** dialog box opens, and the File Linking tab is displayed.

   For an InstallScript-based project: right-click a component and click **File Linking**. The Link Type dialog box opens.
5. Define the dynamic link and click **OK**.

Tip • *You can also use the Components view to add dynamic file links. For more information, see Adding Dynamic File Links to Components.*

Adding Dynamic File Links to Components

You can use the Components view to add dynamic file links to components in both Windows Installer–based and InstallScript-based projects. The procedure for creating dynamic file links differs, depending on which project type you are using.

Windows Installer–Based Projects

The following procedure applies to Basic MSI, DIM, InstallScript MSI, and Merge Module projects.

Task ***To add a dynamic link to a component:***

1. In the View List under **Organization**, click **Components**.
2. In the **Components** explorer, expand the component for which you want to add dynamic file links, and then click the **Files** item under that component.
3. Right-click the **Files** pane and click **Dynamic File Linking**. The **Modify Dynamic File Links** dialog box opens.
4. Click the **New Link** button. The Dynamic File Link Settings dialog box opens. This dialog box is where you set the source folder for your link, indicate whether to include its subfolders and whether the files are self-registering, and specify which file types to include and exclude.
5. Click **OK**. Any file conflicts are resolved with prompts to overwrite existing files with the new versions. Click **Yes** to overwrite, **No** to retain the existing file version, or **Cancel** to exit the dialog box without saving any of your dynamic file link settings. You will also be warned if the folder contains no files, or if it is already in use in another dynamic link.

Your new file link details appear in the Files pane.

InstallScript-Based Projects

The following procedure applies to InstallScript and InstallScript Object projects.



Task ***To add a dynamic link to a component:***

1. In the View List under **Organization**, click **Components**.
2. In the **Components** explorer, click the component that should contain the dynamic link.
3. Select the value of the **Link Type** property and click the ellipsis button (**...**). The Link Type dialog box opens.
4. Specify the folder that contains the files to include, and specify any other desired options.
5. Click **OK**.



Tip *• You can also use the Files and Folders view to add dynamic file links. For more information, see Creating a Dynamic Link.*

Setting a Key File for a Dynamic File Link



Project *• This information applies to dynamic links that use the one-component-per-directory method of component creation (as described in Determining the Appropriate Component Creation Method for Dynamically Linked Files) in the following Windows Installer–based projects:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

This information does not apply to dynamic links that use the best practice method of component creation.

A dynamically linked file whose component uses the one-component-per-directory method of component creation cannot be the key file of a component.

To set a key file for a component containing a dynamic link, add a static link to the desired file, and then set the static link to be the key file of the component. In the dynamic link settings, enter the full name of the key file in the **Exclude files with the following extensions** field.

For example, you might have a source directory containing several .txt files, and you want the file called Key.txt to be the key file. First, add a static link to Key.txt to a component, and set Key.txt as the key file. Next, create a dynamic link with the same source folder, setting the **Include files with the following extensions** setting to ***.txt** and the **Exclude files with the following extensions** setting to **Key.txt**.

For a dynamic link that includes subdirectories, the build process sets the first file in a subdirectory's component as the key file of the dynamically generated component.

Overwriting Files and Components on the Target System



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

At run time, Windows Installer considers the following questions when determining whether files should be overwritten on the target system:

- Should the component be installed?
- If the component should be installed, do its files need to be installed or updated?

The following sections explain how Windows Installer evaluates these questions.

Determining Whether a Component Needs to Be Installed

During costing, Windows Installer checks the key path of a component to determine if the component should be installed. If the component has a directory or ODBC key path, Windows Installer installs the component. In addition, if a registry key path is used for the component and both of the following conditions are true, Windows Installer installs the component:

- No is selected for the Never Overwrite setting of the component.
- The key path does not exist on the target system.

If one or both of those conditions are false for a component that has a registry key path, Windows Installer does not install the component at run time.

If a component has a key file instead of a key path, Windows Installer checks the target system for the presence of that file. Windows Installer installs the component in all of the following scenarios:

- The file is not present on the target system.
- The file is present on the target system, and the key file of the component has a higher or equal version number compared to the version number of the file on the target system.
- The file is present on the target system, and the key file of the component and the file on the target system are unversioned.

If the key file of the component has a lower version number than the file on the target system, Windows Installer does not install the component at run time. In addition, Windows Installer does not install the component at run time if the key file of the component is unversioned but the file on the target system is versioned. Thus, Windows Installer does not install a component if its key file could be downgraded at run time.

Determining Whether a File Needs to Be Installed

If Windows Installer determines that a component needs to be installed, it reviews the files for that component and determines if they need to be installed. If the name and target location of the key file matches the name and location of a file on the target system, Windows Installer uses the following rules by default to determine whether the file in your installation should overwrite the corresponding file on the target system:

- **Always Overwrite**—If the Always Overwrite check box for the file is selected, the file on the target system is overwritten, regardless of version number.

- **Versioned Files**—In all cases, the file with the highest version is maintained, even if the file already on the target system has a higher version than the one being installed. Additionally, a file of any version is maintained over unversioned files.
- **File Language**—All other things being equal, the file that is the same language as the installation is maintained over different language versions of the file. The only exception to this rule applies to multiple language files. Files with multiple languages are maintained over single language versions of a file.
- **Date**—If the modified date of a file already present on the target system is later than the creation date of that file, the file is not overwritten. This rule protects user preference files from being overwritten during an upgrade or reinstallation.

If a file is in the component that is being installed but it is not present on the target system, Windows Installer always installs that file.

Note • *The **REINSTALLMODE** property can be set to modify the default file-versioning rules. For more information, see Understanding File Overwrite Rules.*

Displaying Predefined Folders in the Files View

Task ***To display a predefined destination folder:***

1. In the View list under **Specify Application Data**, click **Files**.
2. In the **Destination computer's folders** pane, right-click **Destination Computer**, point to **Show Predefined Folders**, and then click the type of predefined folder that you want to use.

Companion Files

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The use of companion files enables you to bind the installation action of one file to another file. For example, if your installation project has two files—**FileA.exe** and **FileB.dat**—companion files let you bind **FileB.dat** to **FileA.exe** so that if **FileA.exe** needs to be installed or reinstalled, then **FileB.dat** is also installed or reinstalled. If **FileA.exe** needs to be uninstalled, then **FileB.dat** is also uninstalled.

Using Companion Files

This mechanism is useful when trying to override the Windows Installer's default file versioning rules. For instance, file versioning rules state that for non-versioned files, any file on the target machine that has a modified date later than its created date is considered user data and should not be overwritten. This is not always a valid assumption, so you may want to use companion files to bind a non-versioned file to a versioned file.

Creating Companion File Associations

Task ***To create a companion file association:***

1. In the **Files and Folders** view, add the versioned file to your project.
2. Open the **Components** view.
3. In the **Components** explorer, expand the component that contains the file that you just added, and then click **Files**. Note the value in the **Key** column for that file.
4. Right-click the file and then click **Add**.
5. Select the unversioned file to add it to the selected component.
6. Right-click the second file and then click **Properties**. The Properties dialog box opens.
7. Select the **Override system version** check box.
8. In the **Version** box, type the name of the value noted in the **Key** column from step 3.

Note • *InstallShield generates a unique file key every time you add a file. Therefore, if you remove the file that you have created a binding to, it is possible that the file key will not persist. You would need to update the version of the unversioned file.*

Configuring Permissions for Files and Folders

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield lets you configure settings for securing files and folders for end users who run your product in a locked-down environment. You can assign permissions for a file or folder to specific groups and users. For example, you may assign Read, Write, and Delete permissions for a particular file to the Administrators group, but only Read permissions for all of the users in a different group.

Task ***To configure the permissions for a file or folder:***

1. In the View List under **Application Data**, click **Files and Folders**.
2. For a file: In the **Destination computer's files** pane, right-click the file and then click **Properties**. The **Properties** dialog box opens.

   For a folder: In the **Destination computer's folders** pane, right-click the folder and then click **Properties**. The **Properties** dialog box opens.
3. Click the **Permissions** button. The **Permissions** dialog box opens.
4. Add, modify, and remove permissions entries as needed. For more information, see Permissions Dialog Boxes for Files and Directories.

Depending on what is selected for the Locked-Down Permissions setting in the General Information view of your project, InstallShield adds permissions data to either the **ISLockPermissions** table or the **LockPermissions** table. To learn more, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.

Tip • *You can also configure permissions for a component's destination folder. To do so, first click a component in the Components view. Next, click the value for the Destination Permissions setting, and then click the ellipsis button (...). The Permissions dialog box opens. On this dialog box, you can configure permissions as needed.*

Extracting COM Data When Files Are Added

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

When you drag and drop new files in the Files and Folders view or in the Files subview under a component, you can statically extract COM data for self-registering files. This is an alternative to extracting COM registration data at build time.

This option is also available in the COM Registration node under Advanced Settings in the Components view.

Extracting a File's COM Data

Task ***To extract COM data on demand, rather than at build time:***

Right-click the file (or in the **COM Registration** node, right-click **COM Registration**) and click **Extract COM Data for Key File**.

Note *• This option is enabled only if the file is an .exe file or a file that is self-registering, and if the file is the component's key file.*

Tip *• If you are using InstallShield on a 64-bit system, InstallShield can extract COM data from a 64-bit COM server. In order to install the data to the correct locations, the component must be marked as 64 bit. To learn more about 64-bit support, see Targeting 64-Bit Operating Systems.*

Refreshing a File's COM Data

If the file's COM data has already been statically extracted, you can select Refresh COM Data for Key File. When you refresh the COM data, the old COM data is deleted from the project, then the current COM data for the file is added to the project.

Installing Self-Registering Files

The installer uses "traditional" self-registration to register your self-registering files when the files are marked as self-registering and (in a Windows Installer–based project) they are in a component that has the COM Extract at Build property set to No. The files are unregistered when they are uninstalled. The self-registration functions supported are **DllRegisterServer** and **DllUnregisterServer**.

Before setting a file's Registration property, make sure the file is self-registering. According to Setup Best Practices, there should be only one .dll file, .ocx file, or .exe file per component.

Tip *• The recommended method for installing self-registering files with Windows Installer is to write the registration information to the .msi database tables (Class, ProgID, and others). Instead of marking a file as self-registering, you can use the component's advanced settings, extract the COM information at build time, or extract the COM information when you add a file in the Files and Folders view in order to register the ProgIDs, type libraries, and so on. Using the advanced settings works whether you are installing the file in the installation or advertising it for "just-in-time installation." A further advantage is that the file can be unregistered if the installation fails.*

Even if you set the component's COM Extract at Build property to No, any information that is contained in the COM Registration advanced setting will nonetheless be registered during installation. You might want to check the COM Registration advanced setting and the component's registry data to verify that the entries are intentional and do not conflict with those made by the self-registration functions.

Identifying Properties and Dependencies of .NET Assemblies

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

- *Merge Module*

InstallShield offers support for identifying the properties and dependencies of .NET assemblies. The functionality depends on the project type that you are using.

Windows Installer–Based Projects

In Basic MSI, DIM, InstallScript MSI, and Merge Module projects, when you add a .NET assembly to your project, it is recommended that you set it as the key file of its component. No other .NET assemblies should be included in that component.

When a .NET assembly is the key file of its component, you can use one of the following methods to identify its dependencies:

- If you know which files and merge modules are required for the .NET assembly, you can manually add them to the same feature that contains the .NET assembly component in your project. This method is recommended, since it gives you the most control over what is included in your project.
- If you do not know which files and merge modules are required for the .NET assembly, use one of the following built-in methods:
  - To identify the .NET assembly's dependencies on demand, use the Static Scanning Wizard to detect possible dependencies. This wizard displays a list of the dependencies that it finds, and it lets you specify whether you want to include each one in your project. Note that this method is available in Basic MSI and InstallScript MSI projects, but not in DIM or Merge Module projects.
  - To identify the .NET assembly's dependencies each time that you build your project, select Dependencies and Properties for the component's .NET Scan at Build setting. If InstallShield detects any possible missing dependencies at build time, InstallShield incorporates them into the release that it generates.

Configuring the properties of a .NET assembly makes it possible for Windows Installer to update and uninstall the assembly when needed. When a .NET assembly is the key file of its component, you can use either of the following methods to identify its properties:

- Manually enter the properties and their values: In the Components view, under the Advanced Settings area for the component that contains the .NET assembly, select .NET assembly subnode under the Assembly node, and configure the properties as needed. Note that the properties and values that you enter must match the information in the assembly's manifest file. If they do not match, the assembly might be left on the target system when the feature that contains the .NET component is uninstalled.
- To identify the .NET assembly's properties each time that you build your project, select the Properties Only option or the Dependencies and Properties option for the component's .NET Scan at Build setting. If InstallShield detects any possible missing properties at build time, InstallShield incorporates them into the release that it generates.

InstallScript Projects

In InstallScript projects, when you add a .NET assembly to a component in your project, you can use the component's .NET Assembly setting to identify the assembly as a local .NET assembly. You can also use one of the following methods for identifying the dependencies of the .NET assembly:

- If you know which files and merge modules are required for the .NET assembly, you can manually add them to the same feature that contains the .NET assembly component in your project. This method is recommended, since it gives you the most control over what is included in your project.

- If you do not know which files and merge modules are required for the .NET assembly, use one of the following built-in methods:
  - To identify the .NET assembly's dependencies on demand, use the Static Scanning Wizard to detect possible dependencies. This wizard displays a list of the dependencies that it finds, and it lets you specify whether you want to include each one in your project.
  - To identify the .NET assembly's dependencies each time that you build your project, select Dependencies for the component's .NET Scan at Build setting. If InstallShield detects any possible missing dependencies at build time, InstallShield incorporates them into the release that it generates.

If Not an Assembly is selected for the component's .NET Assembly setting, InstallShield does not scan for dependencies of the component's assembly.

Note that when Local Assembly is selected for the component's .NET Assembly setting, the installation performs COM interop registration and configures .NET installer class information for the component at run time.

Scanning 64-Bit .NET Assemblies for Dependencies



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

If you use InstallShield on a 64-bit version of Windows Vista or later or a 64-bit version of Windows Server 2008 or later, and you use either of the following built-in methods for detecting dependencies, InstallShield can scan for 64-bit dependencies of the 64-bit .NET assemblies in your project:

- To identify a 64-bit .NET assembly's dependencies on demand, use the Static Scanning Wizard to detect possible dependencies. This wizard displays a list of the dependencies that it finds, and it lets you specify whether you want to include each one in your project.
- To identify a 64-bit .NET assembly's dependencies each time that you build your project, select one of the dependency options for the component's .NET Scan at Build setting. For InstallScript projects, the component's .NET Assembly setting must also be set to Local Assembly. If InstallShield detects any possible missing dependencies at build time, InstallShield incorporates them into the release that it generates.

These methods also scan for 32-bit dependencies of the 32-bit .NET assemblies in your project.

Note that if you use InstallShield on a 32-bit version of Windows, these built-in scans can check for only 32-bit dependencies of the 32-bit files in your project. If your project includes 64-bit files, you can manually add any dependencies to the project as needed.

How the Built-In Scanning Methods Detect Platform-Specific and Platform-Independent .NET Dependencies in Windows Installer–Based Projects

In Basic MSI, DIM, InstallScript MSI, and Merge Module projects, when you use either of the aforementioned built-in methods for detecting .NET dependencies, InstallShield uses a specific order for detection, as indicated in the following table.

Table 4-3 • Order for Detecting .NET Assembly Dependencies

Type of File Being Scanned	Scan Results on 32-Bit Windows Systems	Scan Results on 64-Bit Windows Systems
Platform-independent .NET assembly in the GAC	InstallShield uses the following order when scanning for dependencies: **1.** 32-bit-specific .NET dependencies **2.** Platform-independent .NET dependencies	InstallShield uses the following order when scanning for dependencies: **1.** 64-bit-specific .NET dependencies **2.** Platform-independent .NET dependencies
64-bit .NET assembly in the GAC	InstallShield does not detect any dependencies.	InstallShield uses the following order when scanning for dependencies: **1.** 64-bit-specific .NET dependencies **2.** Platform-independent .NET dependencies
32-bit .NET assembly in the GAC	InstallShield uses the following order when scanning for dependencies: **1.** 32-bit-specific .NET dependencies **2.** Platform-independent .NET dependencies	InstallShield uses the following order when scanning for dependencies: **1.** 32-bit-specific .NET dependencies **2.** Platform-independent .NET dependencies

If a .NET assembly is not in the GAC, InstallShield checks the same folder that contains the assembly when scanning for dependencies, and then it checks any subfolders.

Reviewing .NET Dependency Scanner Results



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*
- *Merge Module*

InstallShield offers different built-in methods for identifying dependencies of .NET assemblies. If you use these built-in methods, the scanning may identify as a dependency a file or merge module that is not required by your product. If you are using the Static Scanning Wizard and that occurs, you can specify in the wizard that you do not want to add that dependency to your project. If you are using the .NET scan at build functionality to automatically include any possible .NET dependencies in your project at build time, you can change the values of the component's settings so that the .NET assembly is not scanned for dependencies automatically:

- In Basic MSI, InstallScript MSI, and Merge Module projects, InstallShield scans each component's .NET assembly for dependencies automatically at build time by default. You can override this default behavior by changing the value of a component's .NET Scan at Build setting as needed.
- In InstallScript projects, InstallShield does not scan each component's .NET assembly for dependencies at build time by default. You can override this default behavior by changing the values of a component's .NET Assembly setting and its .NET Scan at Build setting as needed.

In addition, InstallShield enables you to specify on a machine-wide basis any files that you want to be included or excluded automatically any time that you perform a dependency scan through InstallShield. For more information, see Filtering Files in Dependency Scanners.

To obtain the best results when you are trying to identify dependencies, it is recommended that you thoroughly test your product and its installation on a clean machine. If your product does not behave as expected, determine whether any dependencies are missing from the machine, and if so, whether they should be included in the installation.

Installing Fonts Through InstallScript and InstallScript Object Projects



***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

When you include a font file—.ttf, .ttc, or .fon file—in your installation project (using either the Components view or the Files and Folders view), the file is automatically marked internally to be registered on the target machine if you have enabled global font registration. At run time, the **OnInstalledFontFile** event handler function is called immediately after each font file is installed; the default code for this event handler function calls the **RegisterFontResource** function to register the font if it is internally marked for registration.

Font Titles

InstallShield determines the font title that the installation registers for the font in the following manner:

- For an .fon file, InstallShield—by default—reads the font title from the registry of the source system. If no title can be found in the registry, InstallShield stores the file name as the font title. For an .fon file in a component whose Link Type property is set to Static, InstallShield displays the font title in the Font title box on the file's File Properties dialog box.
- For a TrueType file (.ttf or .ttc file), by default the installation reads the font title from the file at run time, and at design time no text is displayed in the Font title box.

- For all three types of files, in a component whose Link Type property is set to Static, if you enter non-default text in the Font title box, the installation registers that text as the font title.

Installing Fonts to the Windows Fonts Folder



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*



Task ***To install a font file to the Windows Fonts folder, do either of the following:***

- In the **Files and Folders** view, place the font file in the **Destination computer's folders** pane's **Fonts Folder** folder, which is a subfolder of the **Windows** folder.
- In the **Components** view, place the font file in a component whose **Destination** property is set to **<FOLDER_FONTS>**.

Enabling or Disabling Global Font Registration



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

You can enable or disable automatic registration of font files globally.



Task ***To enable or disable global font registration:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Font Registration** setting, specify the appropriate option:
   - To enable global font registration, select **Enabled**.
   - To disable global font registration, select **Disabled**.

Enabling or Disabling Registration of a Font File



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

You can enable or disable automatic registration of font files for individual files.

Task ***To enable or disable automatic registration of an individual font file:***

1. Open the **Components** view, **Setup Design** view, or **Files and Folders** view.
2. Right-click the font file and click **Properties**. The **Properties** dialog box opens.
3. To enable automatic registration of the font file, select the **Register Font File** check box.

   To disable automatic registration, clear the **Register Font File** check box.
4. Click **OK**.

Using Components

Components are elements of the application from the installation developer's perspective. Components are not visible to the end user. When the user selects a feature for installation, the installer determines which components are associated with that feature and then those components are installed. Components of an application would contain the executable binary files, data files, shortcuts, help system files, and registry entries.

You can create and modify components in the Setup Design view (for installation projects) or the Components view.

Component-Feature Relationships

Project *• Features are not used in DIM or Merge Module projects.*

Components are associated with features in the Setup Design view. For more information, see Associating New Components with Features.

Files

Expand a component's tree in the Setup Design view or the Components view and click Files to view a list of all files associated with that component. To learn how to add files, see Adding Files to Components.

Setup Best Practices

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield makes adhering to Setup Best Practices easier by alerting you to Best Practices violations while you are creating components in the Setup Design view and in the Component Wizard. By following these guidelines, you can create clean installations that distribute reusable components efficiently and avoid problems that result from file incompatibility.

When you are creating components in the Setup Design view, the Setup Best Practices Wizard monitors the files that you add to the components, alerts you when you have done anything that conflicts with Best Practices, and then gives you the opportunity to correct the action in the wizard. To turn off the wizard's automatic scanning of your setup design, select Options from the Tools menu.

InstallShield checks for compliance with the following Setup Best Practices:

Table 4-4 • Setup Best Practices that InstallShield Monitors

Best Practice	Description
Components should not contain multiple .exe, .dll, .ocx, .chm, or .hlp files	Each component should contain only one portable executable file (an .exe, .dll, or .ocx file) or WinHelp file (.hlp file). Windows Installer components are designed such that all of the advanced settings and component settings, such as the GUID code, refer ideally to a single portable executable file or help file. Place other .exe, .dll, .ocx, and .hlp files into new components. One reason for this rule is that, at run time, if the user chooses Repair for an installed product, Windows Installer checks for the existence of each installed component's key file. If the key file is missing, Windows Installer reinstalls the missing component. Therefore, if a file that is not the key file of a component is missing, it may not be restored during repair mode.
Use merge modules	Merge modules contain all of the files, registry entries, and logic necessary to install a distinct piece of functionality. You should not distribute a file for which a merge module is available. Using merge modules also helps you comply with two related requirements—the Best Practice to avoid associating a file with more than one component and the Windows logo guideline not to ship any core components. If the Best Practices monitoring option is enabled, the Setup Best Practices Wizard alerts you whenever you try adding to a component any file that is part of the merge modules that InstallShield provides.

You should also be aware of the following component creation Setup Best Practices, to which InstallShield does not automatically alert you:

Table 4-5 • Setup Best Practices for which InstallShield Does Not Automatically Provide Alerts

Best Practice	Description
Put a shortcut target in its own component	Any file that serves as the target for a shortcut requires its own component. That file must be the key file for the component.
Group other files into components	Organize all files that do not fall into any of the above categories in components according to the files' requirements, such as a common destination folder or version checking.

Table 4-5 • Setup Best Practices for which InstallShield Does Not Automatically Provide Alerts (cont.)

Best Practice	Description
Do not self-register files	Instead of calling self-registration functions to register and unregister COM server information, you should register this information for the component during installation (and the installer will unregister it during uninstallation). Self-registration is an unreliable method for registering and unregistering files. One advantage of having the installation register the information is that if the file is advertised, or not immediately installed, the registration is in place when the file is later requested from the installer. Therefore, InstallShield supports advanced settings specifically for registering COM servers.

Component Creation

InstallShield offers several methods for creating components.

- Use the Setup Design view (in installation projects only) or the Components view to create a new component and manually configure it. For more information, see Using the Setup Design or Components Views to Create a Component.
- Use the Component Wizard to allow InstallShield to define components for you according to Setup Best Practices. This wizard is available in the following project types: Basic MSI, DIM, InstallScript MSI, and Merge Module. For more information, see Using the Best Practices Option with the Component Wizard.
- If you are creating a component for a COM server, or a font, it is recommended that you use the Component Wizard. This wizard is available in the following project types: Basic MSI, DIM, InstallScript MSI, and Merge Module. For more information, see Using the Component Type Option with the Component Wizard.

Using the Setup Design or Components Views to Create a Component

For installation projects, you can create a new component in the Components view and then associate it later with a feature, or you can create a component under a specific feature in the Setup Design view, as described below. Note that the Setup Design view is not available in DIM or Merge Module projects.

Using the Setup Design View to Create a Component

For installation projects only, follow the steps below to create a component in the Setup Design view.



Task ***To create a component in the Setup Design view:***

1. In the View List under **Organization**, click **Setup Design**.
2. In the **Setup Design** explorer, right-click a feature or subfeature and click **New Component**, or press CRTL+INSERT. InstallShield adds a new component with the default name **New Component n** (where *n* is a successive number).
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.

The new component is automatically associated with the selected feature.

Using the Components View to Create a Component

For installation projects, DIM projects, and Merge Module projects, follow the steps below to create a component in the Components view.

Task ***To create a component in the Components view:***

1. In the View List under **Organization**, click **Components**.
2. Right-click the **Components** explorer and click **New Component**, or press INSERT. InstallShield adds a new component with the default name **New Component n** (where *n* is a successive number).
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.
4. For installation projects, associate the component with one or more features.

Note • *In installation projects, components that are not associated with a feature are displayed with the orphaned component icon ().*

In InstallScript projects, the following characters are invalid in component names:

`\ / : * ? " ' < > |`

When you create a component, its property sheet is displayed to the right. You should immediately specify the component's properties, including the required Component Code (Windows Installer–based projects only) and Destination properties.

After you have created a component and specified its properties, you can associate your application's files, registry entries, and shortcuts with it.

Using the Best Practices Option with the Component Wizard

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The fastest way to create components is to launch the Component Wizard and have InstallShield organize your files into components for you. Select the **Create components for me using best practices** option to have the wizard generate components by applying Setup Best Practices to the files you specify.

Task ***To launch the Component Wizard, do one of the following:***

- For installation projects only: Right-click a feature in the **Setup Design** view and select **Component Wizard**.
- For installation projects, DIM projects, and Merge Module projects: Right-click **Components** in the explorer of the **Components** view and click **Component Wizard**.

Best Practice Rules for Creating Components

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you select the Best Practices option for creating a component through the Component Wizard, the wizard automatically organizes your files into components based on Setup Best Practices. It creates components according to the following rules:

- The wizard looks at every file that you add to see if it is already included in your project. If the file is a duplicate, the wizard does not create a component for it.
- A new component must be created for each portable executable file (.exe, .dll, or .ocx file). The component bears the name of the portable executable file. The wizard creates a GUID for the Component Code property and sets the portable executable file as the component's key file.
- The Component Wizard attempts to extract registration information for each portable executable file. If it succeeds, the wizard creates a COM server component and writes the data to the component's COM Server advanced setting.
- Every help (.hlp) file that you specify will reside in its own component along with its associated contents (.cnt) file. The component is named after its help file. The wizard creates a GUID for the Component Code property and makes the help file the component's key file. The same rule applies to HTML Help (.chm) files and the .chi files that accompany them.
- The Component Wizard puts all font (.ttf and .ttc) files into a single component called Font Files. The wizard creates a GUID for the Component Code property and sets the first file in the list as the key file. Any .fon files that you specify will automatically be included in the AllOtherFiles component that is created. Since .fon files do not have a title, they will not be added to the Font table in the .msi package. To have these files added as fonts, you will need to use the Fonts option in the Component Wizard.
- Any files that do not fall into the above categories are grouped into a single component named All Other Files. The wizard creates a GUID for the Component Code property and sets the first file in the list as the key file.

Letting InstallShield Create a COM Server Component



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you select the **Create components for me using Best Practices** option on the Welcome panel of the Component Wizard, InstallShield automatically creates a separate component for each portable executable file that you specify. The wizard also attempts to extract registration information for each portable executable file.

If it succeeds, all of the registration information that maps to the COM Registration advanced setting is written there. The wizard creates any additional entries, such as the InProcServer32 ThreadingModel value, in the component's registry data.



Note *• If your COM server is an executable, you must put an OLESelfRegister attribute with the value of 1 into the Version section of the component's resource file.*

Letting InstallShield Create Font Components



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you select the **Create components for me using Best Practices** option on the Welcome panel of the Component Wizard, InstallShield automatically creates a separate component for all font (.ttf and .ttc) files that you specify. Any .fon files are added to the AllOtherFiles component that is created due to the fact that these files do not have an embedded font title. To have .fon files added to your setup as fonts, use the Fonts option in the Component Wizard.

The destination folder for the font component is set to the destination folder that you chose in the Component Wizard. After the component is created, you can set its Destination Folder to the Windows Installer folder property FontsFolder (that is, use **[FontsFolder]** as the destination), the usual default location for fonts. Place square brackets around FontsFolder, as you would when specifying **[INSTALLDIR]**.

Task ***If the font was not registered on your system, you must specify the font title by doing the following:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. In the explorer, expand the component that contains the font file and click **Files**. The **Files** view opens.
3. Right-click the font file and click **Properties**. The **Properties** dialog box opens.
4. In the **Font title** box, type a font title in the format **FontName (FontType)**—for example, **Roman (All res)**.
5. Click **OK**.

Using the Component Type Option with the Component Wizard

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you have files with special installation requirements, using the **Let me select a type and define the component myself** option of the Component Wizard is the recommended method for organizing them into components.

Task ***To launch the Component Wizard, do one of the following:***

- For installation projects only: In the **Setup Design** view, right-click a feature and click **Component Wizard**.
- For installation projects, DIM projects, and Merge Module projects: In the **Components** view, right-click the **Components** explorer and click **Component Wizard**.

The **Let me select a type and define the component myself** option of the Component Wizard supports the following component types:

- COM servers
- Font files
- Install services
- Control services

Note • *If you want to create a component that installs, starts, stops, or deletes a service during installation or uninstallation, you can use the Services view. You can also use this view to configure extended service customization options, which are not configurable through the Component Wizard. For more information, see Installing, Controlling, and Configuring Windows Services.*

Installing COM Servers



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Although you can create a component in one of several ways, the recommended method for installing a COM server (.exe, .dll, or .ocx) file in an installation is to create a COM server component in the Component Wizard.

The wizard panels displayed depend on the selections you made in previous panels. For example, the Classes panel is not displayed if the wizard succeeds in extracting the registration information in the COM Server Executable panel.



Tip *• InstallShield includes support for 64-bit COM extraction. For more information, see Targeting 64-Bit Operating Systems.*

Installing Fonts



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Although you can create a component in one of several ways, the easiest way to install a font (a .ttf, .fon, or .ttc file) in an installation is to create a Fonts component in the Component Wizard.

In the Component Wizard, the panels that are displayed depend on the selections made in previous panels. For example, the Add New Fonts panel is not displayed unless you selected **I also want to add fonts not installed on this system** in the Add Installed Fonts panel.

Exporting Components to Other Projects



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



Task ***To save a component to another project:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. Right-click the component and click **Export Components Wizard**.



Tip *• You can also access the Export Components Wizard from the Project menu in InstallShield.*

When you export a component into a project, a copy of this component is added to the specified .ism file, along with all of the component's data, such as its files, shortcuts, registry entries, and advanced settings. Any string entries, properties, and path variables used in the dialog are also copied to your new project.

If the target .ism file already has a component of the same name with different properties, the Resolve Conflict dialog box opens to offer you options for resolving the conflicts.

Deleting a Component in Your Project



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You can permanently delete components in both the Setup Design view and the Components view.



Task ***To delete a component:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. Right-click the component and click **Delete from project**.

The component is permanently deleted from your project.

Component-Feature Associations

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Transform*

A component can be associated with as many features or subfeatures as necessary. For example, a text editor product might have two features—an editor and a spell checker. Both the editor and spell checker have dependencies that are in a System .dll files component. When designing this installation, you should associate the System .dll files component with both the editor and the spell checker features.

You can associate components with features in the Setup Design view. There are two ways to associate components with features in the Setup Design view, depending on whether or not the component already exists. For more information, see Associating New Components with Features and Associating Existing Components with Features.

Note *• Components that are not associated with a feature are displayed with the orphaned component icon:*



Associating New Components with Features

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Transform*

Task ***To associate a new component with a feature:***

1. In the View List under **Organization**, click **Setup Design**.
2. Right-click the feature or subfeature and click **New Component** or **Component Wizard**. InstallShield creates a component and associates it with the selected feature.

Associating Existing Components with Features



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Transform*



Task ***To associate an existing component with a feature:***

1. In the View List under **Organization**, click **Setup Design**.
2. Right-click the feature or subfeature and click **Insert Components**.
3. Select the components you want to associate with this feature from the list of components, and click **OK**.

You can move or copy existing components from one feature to another using a drag-and-drop operation:

- To move an existing component, drag the component from one feature to another.
- To copy an existing component, press CTRL while you drag the component from one feature to another.

Disassociating Components from Features



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Transform*



Task ***To remove a component-feature association:***

1. In the View List under **Organization**, click **Setup Design**.
2. Right-click the component that you want to dissociate from the feature, and click **Remove from feature**.

Even though the component is no longer associated with this feature, it is still present in your project.

Adding Data to a Component

You can use the Components view to add files, registry data, and shortcuts to components.

Adding Files to Components

All of the files in a component must share the same component settings.

Criteria for Adding Files to a Component

You can add a group of files to a component only if they satisfy all of the following criteria:

- All files have the same destination folder.
- All files should be installed with the same conditions (including operating system and language).

Create new components organized around your files' installation needs and component properties. You should also be aware of Setup Best Practices when adding files to a component. Setup Best Practices helps you to write clean installations and distribute reusable components effectively.

Methods for Adding Files to Components

The way in which you add files to a component varies slightly depending on how you create your component. The sections below discuss how to associate files with components based on the way in which the component was created.

Creating a New Component in InstallShield

When you create a component by right-clicking in the Setup Design view (for installation projects) or the Components view, first select the component's Files item to view its file list.



Task ***Use one of the following methods to associate files with this component:***

- Drag and drop files from Windows Explorer onto the file list.
- Right-click in the file list and click **Add**. In the resulting dialog box, browse to select as many files as you want to add from that folder. Click **Open**.

Creating a New Component Using the Component Wizard

If you use the Component Wizard to create a component, you can add files to the component in the Files panel.



Task ***Use one of the following methods to associate files with this component:***

- Drag and drop files from Windows Explorer onto the file list.
- Right-click in the file list and click **Add**. In the resulting dialog box, browse to select as many files as you want to add from that folder. Click **Open**.

InstallShield creates links to your application's files. These links are used to locate the files when you build a release of your installation. If any of these links becomes invalid, File Not Found appears next to the file.

All of the above methods for adding files link the files statically, which means that the file links do not change if new files are added to or removed from the source folder. For an alternative to static linking, see Adding Dynamic File Links to Components.

Changing the Value in the Link To Column for a Component's File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

In the Components view, you can expand a component to display the Files node, which shows the files associated with that component. In the Files explorer, the Link To column provides the directory or the path variable that you used when you added the file. You can use the Direct Editor to change what appears in this column.



Task ***To edit the directory:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, click **File**.
3. Locate the file and edit the value in the **ISBuildSourcePath** column.

Deleting a File from a Component



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Task ***To delete a file from a component:***

1. In the View List under **Organization**, click **Components**.
2. In the **Components** explorer, expand the component that contains the file that you want to delete, and then click **Files** item under it. The files associated with that component are displayed in the right pane.
3. Right-click the file that you want to delete and click **Delete**.

Adding Registry Data to Components

You can use the Registry view to add registry keys and values to a component. This information is written to the target system's registry if the component is installed. To learn how to add registry keys and values, see the following:

- Creating Registry Keys
- Dragging and Dropping Registry Entries to Create Registry Keys
- Creating Registry Values
- Importing Registry Files
- Exporting Registry Files

Creating Shortcuts in the Components View

Project • *For installation projects, you can create shortcuts in the Shortcuts view.*

Before you can create a shortcut, you must create a component that contains the file to which the shortcut is going to link.

Task ***To create a new shortcut:***

1. In the View List under **Organization**, click **Components**.
2. In the **Components** explorer, expand the component that should be associated with the shortcut that you are creating, and then click **Shortcuts**. The **Shortcuts** explorer opens in a new pane.
3. In the **Shortcuts** explorer, right-click a destination folder and click either **New Folder** or **New Shortcut**. You can create a program folder if you want your shortcut to appear under your company name, for example.

   If you created a folder for your shortcut, create the shortcut by right-clicking your new folder and clicking **New Shortcut**.
4. Define the shortcut's properties.

Adding Subfolders to Statically Linked Components

Project *• This information applies to InstallScript projects.*

Task ***To add a subfolder to a statically linked component:***

In the **Components** or **Setup Design** view, right-click the component's **Static File Links** subnode and then click **New Folder**.

Component Key Files

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

One of your component's files can serve as its key file. A key file is a file that the Windows Installer uses to detect the component's presence on the target machine and determine whether it needs to be updated. In order to create advanced component settings or shortcuts, a key file must be specified.

Caution *• If your project includes dynamically linked files, InstallShield may automatically set some of the dynamically linked files as the key file of a component. For more information, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.*

Setting Component Key Files

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Task ***To specify one of your component's files as the key file:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. In the explorer, expand the component and click **Files**.
3. Right-click in the files list and click **Set Key File**. The file icon ( ) for that file is replaced with a key icon ( ).

A component can have either one key file or one key path. If you have already set a key file or a key path for a component, a warning message box appears if you try to set another key file. Click Yes in the message box to replace the existing key file or key path with the new key file.

Note • *You cannot specify a key file in the Files and Folders view. Key files must be set in either the Setup Design or Components view.*

Clearing a Key File from a Component

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you no longer want a file to serve as the component's key file, you can clear the key file from that component.

Task ***To clear the key file from a component:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. In the explorer, expand the component and click **Files**.
3. Right-click the key file and click **Clear Key File**.

Component Settings

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you create a component, its settings have default values, depending on whether you created the component using a view, or using a wizard.



Task ***To edit a component's settings:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. Click the component that you want to configure, and modify its settings as needed.

For descriptions of each of the settings, see Component Settings.

Component Destination vs. Feature Destination



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Note that DIM, Merge Module, and MSM Database projects do not include features. These types of projects are added to features in Basic MSI projects. Thus, components in DIM, Merge Module, and MSM Database projects are associated with the features in the installation projects that consume them.

Both components and features have a destination setting, but there are some differences in how they are used when an end user runs your installation.

Allowing End Users to Change the Destination

You can use the feature's destination setting if you want to allow the end user to modify where the feature is installed on the target system. Because a feature's destination can be changed by the end user, you must use a public property for the feature destination. A public property has a name with only uppercase letters (for example, **INSTALLDIR**).



Note *• If you want all the components in a feature to be installed to the feature's destination, you must set all of the components' destinations to match the feature's destination.*

Setting a Specific Destination

If you want a component to go to a specific location that cannot be changed by the end user, you must set the component destination to some directory that is not used as a feature destination.

Setting a Destination Folder for the Component's Files



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Each component can have a different destination location for its files. The default value for the Destination setting is as follows:

- For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, and MSM Database, and Transform projects—**INSTALLDIR**, initialized to **[ProgramFilesFolder]Company Name\Product Name** in the product's **INSTALLDIR** property
- For InstallScript and InstallScript Object projects—**TARGETDIR**, initialized by default to **PROGRAMFILES ^ IFX_COMPANY_NAME ^ IFX_PRODUCT_NAME** in the default script code for the **OnFirstUIBefore** event handler function



Windows Logo *• According to Windows logo requirements, the default destination of your application's files must be a subfolder of the end user's Program Files folder. If you use* ***INSTALLDIR*** *or* ***ProgramFilesFolder*** *as the parent folder for your feature's Destination setting, your files will be installed to the correct location.*



Task ***To change the component's destination folder:***

1. In the View List under **Organization**, click **Setup Design** (installation projects only) or **Components**.
2. Select the component whose destination you want to change.
3. In the **Destination** setting, select one of the options from the list, or click the ellipsis button (**...**) to select or create a directory.

Other Destination Folder Considerations



Project *• These considerations apply to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Component's Remote Installation Setting

Setting the component's Remote Installation setting to Favor Source (or to Optional when the component's feature is set to Favor Source) means that the component's files will not be installed on the target system, regardless of the component's Destination setting.

Feature's Remote Installation Setting

Each feature also has a Destination setting. If different, the component's Destination setting overrides the feature's destination. The feature's Destination setting is optional, but the component's is required.

INSTALLDIR as Default Destination

The assumption behind using **INSTALLDIR** as the default Destination setting for all of your features and components is that you want all of your application's files installed to the same root folder. This way, when an end user changes the destination folder for any of the features in the CustomSetup dialog, the destination folders for all of the features that are set to be installed to the path contained in **INSTALLDIR** also change.



***Note** • If the component's destination is set to something other than **INSTALLDIR**, the components are installed to the destination specified for each component and changing **INSTALLDIR** has no effect on the components' destinations.*

*An installer folder property such as **INSTALLDIR** specifies a default value. An end user can change this value by setting a property when launching **Msiexec.exe** at the command line or by selecting a new destination folder for a feature in the CustomSetup dialog.*

[GlobalAssemblyCache] for Components Containing a .NET Assembly

If a component contains a .NET assembly, you can set the component's destination to **[GlobalAssemblyCache]**. When a .NET assembly is installed to a target system's Global Assembly Cache (GAC), the assembly can be accessed by other applications on the system.

In general, it is preferable to install assemblies to the local application directory. This increases application isolation.



***Note** • To install an assembly to the Global Assembly Cache, the .NET Scan at Build setting for the assembly's component must be set to either Properties Only or Dependencies and Properties.*

Specifying a Component's Destination from the Script



***Project** • This information applies to InstallScript projects.*

Specifying the target destination of a component's files from the script lets you change that destination at run time based on end-user input or other conditions.



Task ***To specify the target destination of a component's files from the script:***

1. In the View List under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, right-click **Script-defined Folders** and click **New Folder**. InstallShield adds a subfolder with the default name **<NEW VARIABLE N>**, where *N* is a successive number. You can rename this subfolder to any string that is enclosed in angle brackets; for example, **<My Script-defined Folder>**.
3. To specify the target destination of an existing component's files:
   a. In the View List under **Organization**, click **Components** or **Setup Design**.
   b. Select the component that you want to configure.
   c. In the component's property grid, click the value of the **Destination** property and in the list, select the name of the subfolder that you created in step 1.

   To create a new component and specify the target destination of its files:
   a. In the **Files and Folders** view, drag the desired file or files from the **Source computer's folders** pane and drop them on the subfolder that you created in step 1. InstallShield creates a new component with the default name **FilesN**.
   b. In your script, call **FeatureSetTarget** to assign a target location to the subfolder that you created in step 1; for example:

```
AskDestPath( "" , "Choose a location for the XYZ files." , svEndUserSelectedPath, 0 );
FeatureSetTarget( MEDIA , "<My Script-defined Folder>" , svEndUserSelectedPath);
```

Uppercase Directory Identifiers and Component Destinations



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Declaring a Directory Identifier in uppercase letters makes that value a public property that can be set from the user interface. In order to allow the end user or administrator to change the destination via the user interface or from the command line, the Directory Identifier for the component's destination must be a public property.

Using mixed-case or lowercase letters for the Directory Identifier defines the directory entry as a private property. Private properties cannot be changed from the user interface.

Specifying Whether a Component's Files and Other Associated Data Are Uninstalled During Uninstallation



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You may need to ensure that a component's files, registry entries, shortcuts, and other data are not uninstalled when end users uninstall your product. Following are examples of scenarios where it may be appropriate to ensure that a component's data are not uninstalled:

- You want to prevent the uninstallation of files that may be used by other products.
- The component contains a font.
- The component is being installed to [SystemFolder]. According to validation rules, if a component is installed to this location, it should not be uninstalled during uninstallation.

The setting that controls whether the component's data are uninstalled depends on which project type you are using.



Task ***To specify whether a component's files, registry entries, and other data are uninstalled:***

1. In the **Setup Design** view (installation projects only) or the **Components** view, select the component that you want to configure. InstallShield displays its settings in the right pane.
2. In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—Configure the **Permanent** setting as appropriate:
   - If the component data should not be uninstalled when end users uninstall the product, select **Yes**.
   - If the component data should be uninstalled, select **No**.

   In InstallScript and InstallScript Object projects—Configure the **Uninstall** setting as appropriate:
   - If the component data should not be uninstalled when end users uninstall the product, select **No**.
   - If the component data should be uninstalled, select **Yes**.

The Permanent and Uninstall settings apply to all types of data that are associated with a component. This includes files, registry entries, shortcuts, XML file changes, and SQL scripts.

Configuring Component Conditions

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Conditional installation of your components can be useful if you are creating different versions of the same product—for example, a trial version and a full version. You might not want to provide full functionality in the trial version, therefore you would not install all of the components. Another use for conditional component installation is to save disk space. If the target system does not have enough disk space for all of the components, you can set non-required components to install conditionally.

Using the component's Condition setting, you can enter a statement that the installation must evaluate before setting up your component's data on the target system. The component is not installed if its condition evaluates to false. However, the component is installed or advertised if its condition evaluates to true, assuming that its feature is selected for installation.

Task ***To set a condition for a component in your project:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. Select the component that you want to configure.
3. Click the **Condition** setting and then click the ellipsis button (**...**). The **Condition Builder** dialog box opens.
4. Do one of the following:
   - In the **Conditions** box, type the component condition.
   - Use the **Properties** list, the **Operators** list, and the **Add** buttons to build your condition:
     a. In the **Properties** list, select a property and then click the **Add** button. InstallShield adds the property to the **Condition** column.
     b. If your conditional statement should contain an operator, select an operator in the **Operators** list and then click the **Add** button. InstallShield adds the operator to the conditional statement.
     c. If your conditional statement should contain a value, enter the value.
5. Click **OK**.

Important *• InstallShield performs basic condition validation; however, you should still double-check that your condition statements evaluate to the expected outcome. For more information and example conditions, see Building Conditional Statements.*

Reevaluating Component Conditions During Reinstallation



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Windows Installer evaluates a component's condition again when a product is reinstalled if you select Yes for the component's Reevaluate Condition setting. If the condition evaluates to true upon reinstallation, the component is reinstalled—or installed for the first time if the condition initially evaluated to false or if the component was never selected for installation.

Transitive Components

Components that were installed but whose conditions evaluate to false upon reinstallation are removed. Because of this special feature, which allows you, in effect, to swap components during reinstallation, components with Yes selected for the Reevaluate Condition setting are considered *transitive components*.

Consider an application that requires a different .dll file depending on whether it is installed on Windows XP or Windows Vista. You could create a component for each file and attach a condition to each component to check the version of the operating system. The component for Windows Vista would have the following condition:

```
VersionNT=600
```

The Windows XP–specific component might have the following condition:

```
VersionNT<600
```

When the product is installed on a Windows XP system, the appropriate version of the .dll file is installed and the Windows Vista version is not. What happens if the end user upgrades to Windows Vista? When the product is reinstalled, the Windows XP–specific component is uninstalled, and the Windows Vista–specific component is installed (as long as these components are both transitive).

Managing Reference Counts for Shared Files



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*

- *MSM Database*
- *Transform*

A file's reference count (also referred to as *refcount*) is the number of products on a target system that use the file. Reference counts help to ensure that if multiple products are sharing a file, the file remains on the target system until all of the products that share it are removed.

Reference counts for shared files are stored in the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs

Both Windows Installer–based projects and InstallScript-based projects include support for managing the reference counts for shared files. The functionality is slightly different, depending on the project type.

Behavior in Windows Installer–Based Installations

In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects, you can indicate that a key file is shared by selecting Yes for the Shared setting of the key file's component. If the installation installs the key file and Yes is selected for the component's Shared setting, a reference count is created in the registry—if it does not already exist—and incremented. During uninstallation of the key file, the reference count is decremented.



***Windows Logo** • Windows logo guidelines require that the reference count be incremented when installing shared files and decremented when uninstalling. Core component files (which they recommend you not install) should not be reference counted.*

Behavior in InstallScript-Based Installations

In InstallScript and InstallScript Object projects, you can mark a component as shared. If the installation installs the component's files and Yes is selected for the component's Shared setting, a reference count for each file is created in the registry—if it does not already exist—and incremented. During uninstallation of the component, the reference count is decremented.

You can call the InstallScript function **GetFileInfo** with the FILE_SHARED_COUNT constant to determine an existing file's reference count.

How to Mark a Component as Shared



Task ***To specify that a component's key file (in a Windows Installer–based project) or that a component (in an InstallScript-based project) is shared:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. Select the component that you want to configure.
3. For the **Shared** setting, select **Yes**.



***Tip** • One example of when you should always mark a component as shared is if its files are to be installed to a shared directory such as the System folder or the Common Files folder.*

Note *• Note that the installation increments any existing reference count for any file in a component regardless of whether you mark the component as shared. However, if no reference count exists, the installation does not create one unless you select Yes for this Shared setting.*

Checking File Versions Before Installing

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

For InstallScript projects, this functionality is handled by the component's Overwrite setting. When you click the ellipsis button (...) in this setting, the Overwrite dialog box opens, enabling you to specify whether files of the component should overwrite existing files on the target system.

The **Never Overwrite** setting for a component enables you to indicate whether you want your installation to overwrite a file if it already exists on the target system:

- If you select Yes, the file—if it exists on the target system—is never overwritten, regardless of the file version. Selecting Yes for this setting overrides file versioning rules.
- If you select No and the file version on the target system is newer than the version being installed, the file on the target system is not overwritten. However, if the version being installed is newer, the file on the target system is overwritten.

Windows Installer checks for the existence of the component's key file when determining if it should install the component. For more information, see Overwriting Files and Components on the Target System.

Installing Files of the Same Name

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The Source Location setting for a component identifies a subfolder where the component's files will be stored in the source disk images, if the component's files are not compressed. The component's files will be copied to this subfolder in your release image.

The Source Location setting does not require a value, and in most cases, it can be left blank. If you enter a value, it must be a valid Windows folder name.

One instance where the Source Location setting could be used is when you are creating an installation that contains more than one language. In this scenario, you can have multiple files with the same name. You can create a component for each language and configure the Source Location setting as needed for each one. When you use the Source Location setting, any file with the same name can be copied onto the disk in two different locations, without the risk of being overwritten.

For example, create two components called *German* and *English*. For the first component's Source Location setting, enter **GermanVersion**. For the second component's Source Location setting, enter **EnglishVersion**. Create two files called *Test.txt*, giving them slightly different contents. Assign each file to a component.

When you build your installation with uncompressed files, two separate folders on the disk images will be created, one called *GermanVersion* and one called *EnglishVersion*. Separate versions of Test.txt will be copied to each of these folders, but neither copy will be overwritten.



Note *• The Source Location setting is not the same as the destination location. While it is conceivable that you might want to copy both versions of the file to the end user's machine, it is more likely that you would want to filter the files by language.*

Setting a Component's Remote Installation Setting



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Remote Installation setting for a component determines whether the component's files are installed on the target system or run from the source medium, such as a CD-ROM or network server. The default value for a new component is Favor Local, which means that the component's files are installed on the target system.



Task ***To change this value so the component's files run only from the source medium:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. Select the component that you want to configure.
3. In the Remote Installation setting, select **Favor Source**. Selecting **Optional** gives this component the **Remote Installation** setting of its feature.

The component's Remote Installation setting overrides the feature's. For more information, see Component's Remote Installation Setting vs. Feature's Remote Installation Setting.



Caution • *If the component contains a Windows service, select the Favor Local option. Although an end user could change the feature's installation state through the CustomSetup dialog, the Windows Installer cannot install a service remotely.*

Component's Remote Installation Setting vs. Feature's Remote Installation Setting



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *Transform*

The component's Remote Installation setting always overrides the feature's Remote Installation setting. For example, if a component's Remote Installation setting is set to Favor Local, its files are installed on the target system regardless of the feature's Remote Installation setting.

The files are run from the source medium when a component's Remote Installation setting is set to Favor Source. If you want a component's feature to dictate whether the files run from the source medium, select Optional for the component's Remote Installation setting.

When a component is associated with more than one feature and Optional is selected for the component's Remote Installation setting, the files are installed locally if any of its features is set to Favor Local. If the component is set to Favor Local or Favor Source, the files are installed accordingly.

Extracting COM Registration Data at Build Time



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

By selecting Yes for a component's COM Extract at Build setting, you indicate that you want InstallShield to scan the component's key file for COM registration data whenever you build a release that contains that component. The extracted information is placed into the release so that Windows Installer registers the COM server when it is installed or advertised. All the necessary registry settings made by the component are extracted when you select Yes for this setting.

Unlike the Component Wizard, the build process does not write the extracted COM information to the project (.ism) file. Instead, it is dynamic, in that it is updated each time that you rebuild. It also means that you can rebuild an existing release through InstallShield or the command line even when you do not have write access to the project file.



Note *• The PATH system variable on the build machine must be set to include the directories of all of the .dll files to which the COM server links; otherwise, the file will fail to register and COM information will not be extracted.*

The build feedback (displayed in the Output window and in the build log files) details the registration information that was extracted.



Tip *• If you are using InstallShield on a 64-bit system, InstallShield can extract COM data from a 64-bit COM server. In order to install the data to the correct locations, the component must be marked as 64 bit. To learn more about 64-bit support, see Targeting 64-Bit Operating Systems.*

Resolving Conflicts

Despite selecting Yes for the COM Extract at Build setting, you can still specify COM information under the component's COM Registration advanced setting and in the component's Registry explorer. The existing information will always be registered when the component is installed.

If entries are detected under COM Registration, InstallShield asks you if you want to delete them if Yes is selected for the COM Extract at Build setting. If any conflicts are found during the build, you receive a warning about the item in the advanced setting that was overwritten with the dynamically acquired data.

Even with these safeguards, you might want to check the COM Registration advanced setting and the component's registry data to verify that the entries are intentional and do not conflict with the data extracted at build time.

Scanning for .NET Dependencies and Properties



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

InstallShield lets you indicate that a component should be scanned for .NET dependencies and properties at build time. The functionality depends on the project type that you are using.

Windows Installer–Based Projects

In Basic MSI, DIM, InstallScript MSI, and Merge Module projects, you can use the .NET Scan at Build setting to indicate whether you want a component's key file to be scanned for .NET dependencies and properties at build time.



Note *• The build-time scan does not scan the key file of a component if the key file is not a .NET assembly file.*

Several options are available from the .NET Scan at Build setting for a component:

Table 4-6 • Options for the .NET Scan at Build Setting in Windows Installer–Based Projects

Option	Description
None	InstallShield does not scan the key file of this component for .NET dependencies or properties.
Properties Only	At build time, InstallShield scans the key file of this component for .NET properties. InstallShield populates the **MsiAssembly** and **MsiAssemblyName** tables with the assembly properties, as needed.
Dependencies and Properties	At build time, InstallShield scans the key file of this component for .NET dependencies and properties. InstallShield populates the **MsiAssembly** and **MsiAssemblyName** tables with the assembly properties, as needed. In addition, InstallShield adds the missing files, components, and merge modules that are required by the .NET assembly to the release.

Note • *To install an assembly to the Global Assembly Cache, the* ***.****NET Scan at Build setting must be set to either Properties Only or Dependencies and Properties.*

This setting affects how the Static Scanning Wizard scans the files in your project.

InstallScript-Based Projects

In InstallScript and InstallScript Object projects, you can use the .NET Scan at Build setting to indicate whether you want a component's files to be scanned for .NET dependencies at build time.

Several options are available from the .NET Scan at Build setting for a component:

Table 4-7 • Options for the .NET Scan at Build Setting in InstallScript-Based Projects

Option	Description
None	InstallShield does not scan the files of this component for .NET dependencies.
Dependencies	InstallShield scans the files of this component for .NET dependencies. InstallShield adds the missing dependencies to the release.

Specifying the .NET Application File

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The .NET Application File setting is used when the component is scanned at build time (based on the .NET Scan at Build setting) or by the Static Scanning Wizard. The scanner uses this setting—along with the component destination—to determine the value of the File Application setting for the assembly.

The scanning algorithm works in the following way to configure the .NET assembly's File Application setting:

1. The scanning algorithm checks the component's Destination setting. If the value is **[GlobalAssemblyCache]**, the .NET assembly's File Application setting is set to null.
2. If the component's destination is something other than **[GlobalAssemblyCache]**, the scanning algorithm checks the component's .NET Application File setting. If this value is not null, the value of this setting is used to set the assembly's File Application setting.
3. If the component's destination is something other than **[GlobalAssemblyCache]** and the .NET Application File setting is null, the component's key file is used to set the assembly's File Application setting.

Reading Properties Passed to the .NET Installer Class

This is an example of a .NET class that implements the installer class and demonstrates how to read properties that are passed to the installer class from the installation.

```
using System;
using System.Configuration.Install;
using System.Windows.Forms;
using System.Collections;

namespace MyInstall
{
    ///
    /// Summary description for Class1.
    ///
    [System.ComponentModel.RunInstallerAttribute(true)]
    public class MyInstallClass : Installer
    {
        public MyInstallClass()
        {
        }
        public override void Install(IDictionary stateSaver)
        {

            base.Install(stateSaver);
            foreach(string strKey in Context.Parameters.Keys)
            {
                MessageBox.Show(strKey + " is " + Context.Parameters[strKey]);
            }
        }

        public override void Commit(IDictionary stateSaver)
        {
            base.Commit(stateSaver);
            foreach(string strKey in Context.Parameters.Keys)
            {
                MessageBox.Show(strKey + " is " + Context.Parameters[strKey]);
            }
        }

        public override void Uninstall(IDictionary stateSaver)
```

```
        {
            base.Uninstall(stateSaver);
            foreach(string strKey in Context.Parameters.Keys)
            {
                MessageBox.Show(strKey + " is " + Context.Parameters[strKey]);
            }
        }

        public override void Rollback(IDictionary stateSaver)
        {
            base.Rollback(stateSaver);
            foreach(string strKey in Context.Parameters.Keys)
            {
                MessageBox.Show(strKey + " is " + Context.Parameters[strKey]);
            }
        }
    };
}
```

Enabling and Disabling Registry Reflection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *Transform*

Registry reflection keeps the 32-bit registry view and the 64-bit registry view in sync on the target machine.



Note • *Only 64-bit systems with Windows Installer 4 and later support registry reflection. In addition, only Windows Vista and later and Windows Server 2008 and later support it.*

If an end user installs a 64-bit application that has a component with registry reflection enabled, Windows Installer makes the associated registry changes in the 64-bit view of the registry, and the reflector copies the registry changes to the 32-bit registry view. Similarly, if an end user then installs a 32-bit application that modifies the same registry key or value, Windows Installer makes the associated registry changes in the 32-bit view of the registry, and the reflector copies the registry changes to the 64-bit registry view.

If registry reflection is disabled, Windows Installer calls the **RegDisableReflectionKey** function on each key being accessed by the component. This function disables registry reflection for the specified key. Disabling reflection for a key does not affect reflection of any subkeys.

Task ***To enable or disable registry reflection for all new and existing registry keys that are affected by a component:***

1. In the View List under **Organization**, click **Components**.
2. In the **Components** explorer, select the component for which you want to configure the registry reflection setting.
3. To enable registry reflection, set the **Disable Registry Reflection** setting to **No**. This is the default value.

   To disable registry reflection, set the **Disable Registry Reflection** setting to **Yes**.

For more information about registry reflection, see Registry Reflection in the MSDN Library.

Specifying Whether Shared Component Patching Should Be Enabled for a Component

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

InstallShield lets you specify whether you want to enable shared component patching for a component. By default, it is disabled.

If this multiple-package sharing feature were enabled in at least one package that is installed on the target system, Windows Installer 4.5 treats the component as shared among all of those packages. If a patch that shares this component is uninstalled, Windows Installer can continue to share the highest version of the component's files on the system.

The purpose of this multiple-package component sharing is to prevent files from being downgraded during the uninstallation of a patch that contains a component that is shared with one or more other installed packages. The intent is to keep the highest version of the component's files present on the machine after uninstallation of that patch.

Note • *If the DisableSharedComponent policy is set to 1 on a target system, Windows Installer ignores this setting for all packages.*

Windows Installer 4.0 and earlier ignore this setting.

The following diagram illustrates an example of two products, ABC and XYZ, that share a component that contains a file called file.dll. An end user installs product ABC first, and version 1.0.0.0 of file.dll in the shared component is installed. Next, the end user installs product XYZ, which includes version 1.1.0.0 of file.dll. Since this version is higher than the file that was installed for product ABC, Windows Installer overwrites the current version with version 1.1.0.0. Then, the end user installs an uninstallable patch for product ABC. This patch contains version 1.2.0.0 of file.dll. Since this version is higher than the one that is already present on the target system, Windows Installer overwrites the current version with version 1.2.0.0. If the end user uninstalls the patch, either of the following results may occur:

- If the value of the Multiple Package Shared Component setting is Yes for either product ABC or product XYZ and if Windows Installer 4.5 is present, uninstalling the patch from product ABC could restore version 1.1.0.0 to the target system.
- If the value of this setting is No for product ABC and product XYZ, the file would be downgraded to version 1.0.0.0, even though product XYZ used 1.1.0.0. As a result, if product XYZ requires version 1.1.0.0, product XYZ may no longer work properly.

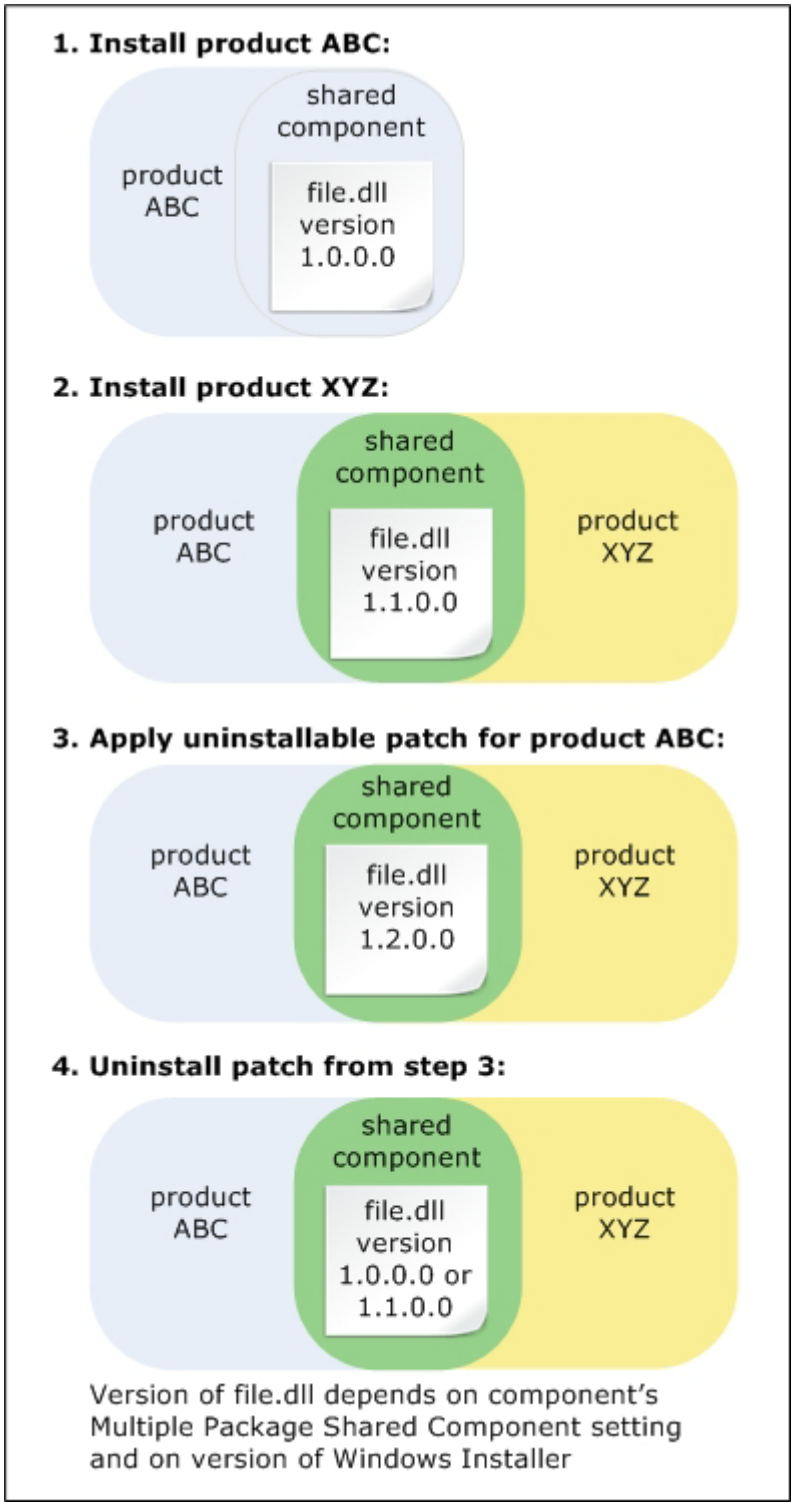


Figure 4-1: If the Multiple Package Shared Component setting for the shared component is set to Yes in either package and if Windows Installer 4.5 is present, version 1.1.0.0 of the file may be restored. Otherwise, the file may be unintentionally downgraded to version 1.0.0.0.

Task ***To specify whether to enable shared component patching for a component:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. Select the component that you want to configure.
3. Select the appropriate value for the **Multiple Package Shared Component** setting:
   - To enable shared component patching, select **Yes**. InstallShield sets the **msidbComponentAttributesShared** attribute of the selected component to indicate that it should be shared.
   - To disable shared component patching, select **No**. InstallShield does not set the **msidbComponentAttributesShared** attribute of the selected component. However, if this component is shared with another package, the **msidbComponentAttributesShared** attribute may be set for this component in that package. Therefore, Windows Installer 4.5 would consider the multiple-package component to be shared.

Advanced Component Settings

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Advanced Settings area under a component in the Components view (and in the Setup Design view) enables you to fulfill installation requirements for special component types. For example, when you copy an .ocx file to the target system, you must register its classes, ProgIDs, and type libraries so that the file's methods can be properly accessed. Advanced settings use Windows Installer's built-in functionality for registering COM servers; setting up ODBC drivers, data sources, and translators; installing, controlling, and configuring Windows services; and registering a file association.

By specifying the advanced settings, you can publish your component and register COM servers, file extension servers, and MIME types. If the component is selected, an advanced setting is made on the target system when the component is installed or advertised. That way, the file is ready to execute once it is installed. Publishing components—a type of advertising—is accomplished through the Publishing advanced setting.

Configuring COM Registration Settings Manually

Project *• This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Important • *The recommended method for installing a COM server is to have Windows Installer register its classes, ProgIDs, and so on. Calling self-registration functions violates Setup Best Practices.*

It is recommended that you avoid using the ***TypeLib*** *table. For more information, see TypeLib Table in the Windows Installer Help Library.*

The easiest way to populate the COM Registration advanced setting is to have the Component Wizard extract the necessary information or extract COM data for a key file. (You can also have it extracted dynamically at build time, in which case you do not need to use this advanced setting.) It is recommended that you modify the COM Registration advanced setting or create COM entries in the component's Registry explorer only if you are familiar with the technical details behind your file's registration.

Task ***To install a COM server solely through editing the component's advanced settings:***

1. Ensure that a single COM server is added to the component. The file must be a single portable executable file (such as an .exe, .dll, or .ocx file), according to Setup Best Practices.
2. Make the COM server the component's key file.
3. For the component's COM Extract at Build setting, select **No**.
4. Expand the component's **Advanced Settings** item to view all of the advanced settings.
5. Click **COM Registration** to configure COM server information.

Right-click one of the items in the COM Registration explorer to modify or to create registration information for your COM server. Right-click an item and click Rename to rename the new item.

Note • *The PATH system variable on the build machine must be set to include the directories of all of the .dll files to which the COM server links; otherwise, the file will fail to register and COM information will not be extracted.*

Configure the settings for each registration item or subitem that you create. More help is available in the help pane in InstallShield when you click each registration setting.

Configuring COM Classes for COM Registration Manually

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



Task ***To register a class ID:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component, and under **Advanced Settings**, select **COM Registration**. The **COM Registration** explorer appears in a separate pane.
3. In the **COM Registration** explorer, right-click **COM Classes** and click **New COM Class**.
4. Type a new name for the COM class if needed. The name that you give the COM class will be registered as the default value under HKEY_CLASSES_ROOT\CLSID\<GUID>. To give the class a new name, right-click it and click **Rename**.
5. Click the new COM class to configure its settings.
6. Specify a context type for this class. The following list tells you which server context is appropriate for which type of COM server:
   - **LocalServer32**—32-bit .exe file
   - **LocalServer**—16-bit .exe file
   - **InprocServer32**—32-bit .dll or .ocx file
   - **InprocServer**—16-bit .dll or .ocx file

If the server context is LocalServer or LocalServer32, click the context to configure the Default Inproc Handler and Argument settings.

Configuring ProgIDs for COM Registration Manually



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

A progID is a string used to identify a class in the format **Component.Class.N**.



Task ***To specify a new programmatic identifier:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component, and under **Advanced Settings**, select **COM Registration**. The **COM Registration** explorer appears in a separate pane.
3. In the **COM Registration** explorer, right-click **ProgIDs** and click **New ProgID**.
4. Type a new name for the ProgID. Use the format **Component.Class.N**. To give the class a new name, right-click it and click **Rename**.
5. Click the new ProgID to configure its settings.

Version-Independent ProgIDs

A version-independent progID is a string used to identify a class in the format **Component.Class**, which is constant for all versions of a class.



Task ***To specify a new version-independent programmatic identifier:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component, and under **Advanced Settings**, select **COM Registration**. The **COM Registration** explorer appears in a separate pane.
3. In the **COM Registration** explorer, right-click **Version-Independent ProgIDs** and click **New ProgID**.
4. Type a new name for the ProgID. Use the format **Component.Class**. To give the class a new name, right-click it and click **Rename**.

Configuring Type Libraries for COM Registration Manually



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



Important • *It is recommended that you avoid using the* ***TypeLib*** *table. For more information, see TypeLib Table in the Windows Installer Help Library.*

Task ***To specify a type library or libID:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component, and under **Advanced Settings**, select **COM Registration**. The **COM Registration** explorer appears in a separate pane.
3. In the **COM Registration** explorer, right-click **Type Libraries** and click **New Type Library**.
4. Type a new name for the type library, or libID, referenced by this COM server. The name that you give the type library item is registered as the default value under HKEY_CLASSES_ROOT\TYPELIB\<libID>\<version>. Use the format **Component.Class**. To give the class a new name, right-click it and click **Rename**.
5. Click on the type library to configure its settings.

Registering a File Extension

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If your application manipulates files with a unique file extension, you can register your file type in the File Types area, which is available as an advanced setting within the Components view and the Setup Design view (for installation projects only). For example, if your application manipulates files with the .xyz extension, registering the file type instructs the operating system to open the file with your application when the user double-clicks its icon.

This advanced setting registers the following information about a file type on the target system when the component is installed or advertised:

Table 4-8 • File Type Information

Information Registered	Description
File Extensions	You can associate file extensions (such as .doc and .txt) with the component's key file.
ProgIDs	By setting the ProgID property in the extension, you can name a ProgID—for example, extfile—that will contain the file type registration.
Verbs	You can register command verbs (such as Open and Print) that appear in the context menu that Windows Explorer displays when an end user right-clicks a file with the current extension.

Table 4-8 • File Type Information (cont.)

Information Registered	Description
MIME Type	You can also register multipurpose Internet mail extension (MIME) types, also known as media types or content types, for the component's key file. You can also associate a MIME type with a class ID.

Note • *File associations are stored in both HKLM\SOFTWARE\Classes and HKCU\SOFTWARE\Classes; you can see a merged view of the data under HKEY_CLASSES_ROOT. It is recommended that you use the File Types editor instead of writing directly to the registry to support Windows Installer feature advertisement.*

Note also that Windows Installer writes file-extension advertisement information to the registry, which appears to be a string of random characters. This behavior is normal.

Installing Assemblies

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Note • *The recommended way to add a .NET assembly to your project is to add an assembly as a component's key file and select Properties Only for the component's .NET Scan at Build setting.*

In the Assembly section of a component's Advanced Settings, you can add a private or global Win32 assembly or .NET assembly to be registered when the current component is installed. Using assemblies helps you install products that are self-contained, without affecting other applications on the system.

Note that a component can contain only one assembly.

Adding Assemblies

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

- *MSI Database*
- *MSM Database*
- *Transform*

When you add a .NET assembly as the key file of a component, InstallShield automatically adds values to the .NET assembly settings when the component is scanned at build time, or when you run the Static Scanning Wizard.

Task ***To add a .NET or Win32 assembly to a component:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.
4. Right-click **Assembly** and then click **New Win32 Assembly** or **New .NET Assembly**.
5. Click the assembly and then configure the **Manifest**, **File Application**, and related settings.

InstallShield adds the information that you enter for the assembly to the **MsiAssembly** and **MsiAssemblyName** tables of your Windows Installer database. You can view and edit these tables using the Direct Editor.

Deleting Assemblies

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Task ***To delete an assembly from a component:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component.
3. Click the **Advanced Settings** item to expand it.
4. Right-click the assembly and click **Delete**.

Testing for .NET Assembly Support on the Target System

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

To see if a target system supports .NET assemblies, you can test the **MsiNetAssemblySupport** property. To see if a target system supports Win32 assemblies, you can test the **MsiWin32AssemblySupport** property. (Win32 assemblies are supported only on Windows XP and later.) The assembly tables, actions, and properties require Windows Installer version 2.0 or later.

Specifying an Application Path for a Component



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The App Paths registry key is a useful installation-related key that helps an executable file find its .dll files without having to modify the PATH environment variable. An executable file's App Paths key looks like this:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths*FileName*.exe

A program's App Paths key typically contains a value named Path, which should contain a semicolon-delimited list of directories where the program's .dll files could be located. Windows uses this key to find your application and its .dll files if their locations are not already in the system's path. If an end user moves or renames your application's executable file through the Explorer shell, Windows automatically updates the file's App Paths key.



Task ***To specify an application path for your component:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. Select the component that you want to configure, and expand its **Advanced Settings** item.
3. Click the **Application Paths** item under **Advanced Settings**.
4. Select the check box of the file for which you would like to create a key.
5. In the **Application Path** column, enter the paths to the file's dependencies, or select a Windows Installer folder property from the list rather than hard-coding a path. Separate multiple paths with a semicolon (;).
6. Click **OK**.

Configuring Device Driver Settings



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

For information on installing device drivers in InstallScript projects, see Installing Device Drivers.

Once you run the Device Driver Wizard, which adds the table and entries, custom actions, features, and components needed to include device drivers in your installation, you can set properties associated with a component that includes a device driver. The following information describes the various options that you can set within InstallShield.

The Device Driver advanced setting's Common tab within the Components view enables you to specify whether the current component includes a device driver and, if so, select desired run-time installation options. The Sequence tab enables you to specify the order in which the project's device drivers (not just the current component's device drivers) should be installed.

Publishing Components



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Publishing advanced setting for a component enables you to specify publishing information for your component. Publishing is a type of advertising (just-in-time installation) in which no user-interface elements are created for the component during installation, but the component can be installed through Add or Remove Programs or when an installed component requests the published component from the installer.

You must create at least one component ID for each advertised component.



Important *• Do not confuse the component ID with the GUID that is entered in the Component Code setting for the component; they must be unique values. The component ID that you use for the Publishing advanced setting is a category identifier that represents the category of components that are being grouped together as a qualified component.*

Each component ID must have at least one qualifier. The qualifier is a string that you can use to distinguish this language or version of the component from any other (for example, to specify a language). It must be unique for the component.

At run time, the installation registers the component IDs and qualifiers and uses these unique values to manage the published components. By calling the Windows Installer functions, your installed component—or another installed component (known as cross-product advertisement)—can request information about an advertised component and install it.

Specifying Publishing Information



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Publishing advanced setting for a component enables you to specify publishing information for your component. Publishing is a type of advertising (just-in-time installation) in which no user-interface elements are created for the component during installation, but the component can be installed through Add or Remove Programs or when an installed component requests the published component from the installer.



Task ***To publish a component that you have created:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. In the explorer, expand the component that you want to publish, and under **Advanced Settings**, select **Publishing**. The **Publishing** explorer appears in a separate pane.
3. Right-click the **Publishing** explorer and then click **New ComponentID** to generate an ID for your new component. InstallShield adds a unique componentID and a corresponding qualifier.
4. In the **Publishing** explorer, click the new qualifier and configure its value.

In the Custom Actions and Sequences view of an installation project, you can set the conditions that need to be fulfilled in order for your product to be advertised on the target system.



Task ***To set the advertisement conditions in an installation project:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Sequences** explorer, expand the **Advertisement** item, and then the **Execute** item.
3. Click the appropriate **Execute** action, and then configure its **Conditions** setting as needed.

Adding ComponentIDs for a Component to Be Published



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You must add at least one componentID to the Publishing advanced setting of a component if you want the component to be published.



Task ***To add a componentID:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component that should have the new componentID, and under **Advanced Settings**, select **Publishing**. The **Publishing** explorer appears in a separate pane.
3. Right-click the **Publishing** explorer and then click **New ComponentID**. InstallShield adds a unique componentID and a corresponding qualifier.

Removing ComponentIDs



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



Task ***To remove a componentID:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component, and under **Advanced Settings**, select **Publishing**. The **Publishing** explorer appears in a separate pane.
3. In the **Publishing** explorer, right-click the componentID and click **Delete**.

Adding Qualifiers to ComponentIDs

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Each componentID must have at least one qualifier. When you create a componentID, by default it has a qualifier. To rename this qualifier, right-click it and click Rename.

Task ***To add a new qualifier:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component, and under **Advanced Settings**, select **Publishing**. The **Publishing** explorer appears in a separate pane.
3. In the **Publishing** explorer, right-click the componentID and click **New Qualifier**. InstallShield creates a new qualifier with a default name.
4. Type a name for the qualifier.

Removing Qualifiers from ComponentIDs

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Task ***To remove a qualifier:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component, and under **Advanced Settings**, select **Publishing**. The **Publishing** explorer appears in a separate pane.

3. In the **Publishing** explorer, right-click the qualifier and click **Delete**.

Configuring Qualifier Settings



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You have the option of specifying an informational string, called *application data*, for each qualifier.



Task ***To set the qualifier properties:***

1. In the View List under **Organization**, click **Setup Design** (for installation projects only) or **Components**.
2. In the explorer, expand the component, and under **Advanced Settings**, select **Publishing**. The **Publishing** explorer appears in a separate pane.
3. In the **Publishing** explorer, click the qualifier.
4. In the **Application Data** setting, enter the appropriate value.

   When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Defining Features

A feature is the smallest installable part of a product, from the end user's perspective. It represents a specific capability of your product—such as its help files or a part of a product suite that can be installed or uninstalled based on the end user's selections. Your entire installation should be divided into features, each of which performs a specific purpose.

Subfeatures are further divisions of a feature. Because features should be self-contained elements of a product or product suite that an end user can selectively install, it might make sense for you to organize portions of your installation as subfeatures of a parent feature.



Tip *• Although you can create many levels of subfeatures, you should keep the design as simple as possible for organizational purposes.*

Creating Features



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Transform*

You can use the Setup Design view or the Features view to create features, as well as subfeatures, for your project.



Task ***To create a feature:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Right-click the top-level item in the explorer and click **New Feature**. InstallShield adds a new feature with the default name **New Feature_n** (where *n* is a successive number).
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.
4. Configure the feature's settings in the right pane.



***Project** • In Basic MSI, InstallScript MSI, MSI Database, and Transform projects, feature names must contain only letters, digits, underscores (_), and periods (.), and they must begin with a letter or an underscore.*

In InstallScript and InstallScript Object projects, the following characters are invalid in feature names:

\ / : * ? " ' < > |



***Tip** • To add a subfeature, right-click the parent feature and click New Feature.*

You can create multiple nested features at one time by adding a new feature and typing **Feature 1\Feature 2\Feature 3** *for the feature's name. InstallShield creates a nested feature structure where* **Feature 3** *is a subfeature of* **Feature 2**, *which is a subfeature of* **Feature 1**.

After you have created all of your product's features and subfeatures, you need to create components and then associate them with your features.

Configuring Feature Settings



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*

- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Transform*

Task ***To configure a feature's settings:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature that you want to configure.
3. Configure the settings in the grid in the right pane.

Setting a Feature's Destination

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Each feature and subfeature can have a different destination location for its files. The default value for a new feature's Destination setting is **INSTALLDIR**, which is set in the General Information view.

Windows Logo *• According to the Windows logo program requirements, the default destination of your product's files must be a subfolder of the Program Files folder or the end user's Application Data folder, regardless of the language of the target system.*

Task ***To change the feature's destination folder:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature whose destination you want to change.
3. In the **Destination** setting, select one of the options from the list, or click the ellipsis button (**...**) to select or create a directory.

   If you want the destination to be configurable at run time, the destination folder that you select must be a public property (containing all uppercase letters).

Note *• You can leave the feature's Destination setting blank.*

Other Destination Folder Considerations

Changing Feature Destinations at Run Time

If you want to change a feature's destination directory at run time based on the end user's feature selection, you can use Windows Installer custom action type 35. For more information about changing a directory's target location, see Changing the Target Location for a Directory in the Windows Installer Help Library.

Feature's Remote Installation Setting

Setting the feature's Remote Installation setting to Favor Source (or to Favor Parent when the subfeature's parent feature is set to Favor Source) means that the feature's files will not be installed on the target system, regardless of the feature's Destination setting.

Component's Destination Setting

Each component also has a Destination setting. If the feature's Destination setting and the component's Destination setting have different values, the component's Destination setting overrides the feature's. The feature's Destination setting is optional, but the component's Destination setting is required.

INSTALLDIR as Default Destination

The assumption behind using **INSTALLDIR** as the default Destination property for all features and components is that all of your application's files should be installed to the same root folder. As a result, when an end user changes the destination folder for any of the features in the CustomSetup dialog, the destination folders for all of the features that are set to be installed to the path contained in **INSTALLDIR** also change.



Note *• If the component's destination is set to something other than* ***INSTALLDIR****, the component is installed to the destination that is specified for the component; changing* ***INSTALLDIR*** *has no effect on the component's destination.*

A directory property such as ***INSTALLDIR*** *specifies a default value. An end user can change this value by setting a property when launching* **Msiexec.exe** *at the command line or by selecting a new destination folder for a feature in the CustomSetup dialog.*

Setting Feature Conditions



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Conditional installation of your features can be useful if you are creating different versions of the same product—for example, a trial version and a full version. You might not want to provide full functionality in the trial version; therefore, you would not install all of the features. Another use for conditional feature installation is to save disk space. If the target system does not have enough disk space for all of the features, you can set non-required features to install conditionally.

Task ***To set a condition for a feature in your project:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature that you want to configure.
3. Click the **Condition** setting and then click the ellipsis button (**...**). The **Feature Condition Builder** dialog box opens.
4. Click the **New Condition** button. InstallShield adds a new condition row to the **Conditions** box.
5. In the **Level** column, type the install level that should be used for the feature if the condition is met.

   Note that if the condition is met, this value overrides the value that is specified for the feature's the Install Level setting.
6. Do one of the following:
   - In the **Condition** column, type the feature condition.
   - Use the **Properties** list, the **Operators** list, and the **Add** buttons to build your condition:
     a. In the **Properties** list, select a property and then click the **Add** button. InstallShield adds the property to the **Condition** column.
     b. If your conditional statement should contain an operator, select an operator in the **Operators** list and then click the **Add** button. InstallShield adds the operator to the conditional statement.
     c. If your conditional statement should contain a value, double-click the condition field, press END so that the insertion point is at the end of the condition statement, and enter the value.
7. Click **OK**.

Important • *InstallShield performs basic condition validation; however, you should still double-check that your condition statements evaluate to the expected outcome. For more information and example conditions, see Building Conditional Statements.*

Run-Time Behavior for Feature Conditions

The default install level of a feature is the value that is configured in the Install Level setting in the Features view or the Setup Design view. This value is overridden if a feature condition evaluates as true; in this case, the install level of the feature is set to the Level value that is associated with the true conditional statement.

Each feature's install level value is compared to the value of the global public property **INSTALLLEVEL**; only those features with install level values less than or equal to **INSTALLLEVEL** are selected for installation.

For example, if you have a feature that you want to be selected by default only if the end user has elevated privileges, you can give the feature a condition of **Not Privileged** and set the install level for that condition to **200**. If the end user does not have elevated privileges, the condition is true, and the feature will be given the install level 200. Because 200 is greater than the default product install level (100), the feature will not be selected by default.



Tip *• You can conditionally hide a feature by giving the feature a condition that sets the install level to the number 0. If the condition is true, the feature will be deselected, and it will not be displayed in the CustomSetup dialog.*

Displaying Features to End Users



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

InstallShield enables you to indicate how you want a feature to be presented to the end user in the custom setup type dialog that corresponds with your project type:

- For Basic MSI, MSI Database, and Transform projects—CustomSetup dialog
- For InstallScript MSI projects—**SdFeatureDialog2**, **SdFeatureMult**, or **SdFeatureTree**

The Display setting for a feature in the Features view or the Setup Design view is where you indicate whether and how the feature should be displayed. Available options are:

Table 4-9 • Options that Are Available for the Display Setting

Option	Description
Visible and Collapsed	The feature is displayed in the run-time dialog with its subfeatures collapsed by default.
Visible and Expanded	The feature is displayed in the run-time dialog with its subfeatures expanded by default.
Not Visible	The feature and subfeatures are not displayed in the run-time dialog.



Note *• Selecting Not Visible for this setting does not have any direct impact on whether a feature is installed. A feature is not automatically installed if it is invisible—it just cannot be deselected if it would otherwise be installed, or selected if it should not be installed.*

Conditionally Selecting Features



Project *• This information applies to the following project types:*

- *Basic MSI*

- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The procedure for conditionally selecting features at run time depends on the project type that you are using.

Basic MSI, MSI Database, and Transform Projects

The Condition setting for a feature enables you to specify a non-default Install Level value if the condition that you specify succeeds. If a feature's Install Level value is less than or equal to the project's **INSTALLLEVEL** property, the feature will be selected to be installed.



Task ***For example, to deselect a feature if the end user does not have administrator privileges:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature that you want to configure.
3. Click the **Condition** setting and then click the ellipsis button (**...**). The **Feature Condition Builder** dialog box opens.
4. Click the **New Condition** button. InstallShield adds a new condition row to the **Conditions** box.
5. In the **Level** column, type **200**.
6. In the **Condition** column, type **Not AdminUser**.
7. Click **OK**.

At run time, if the end user does not have administrator privileges (that is, if the condition succeeds), the Install Level property for the feature is set to 200. Because the default **INSTALLLEVEL** property for a project is 100, the feature is deselected.



Task ***You can change the INSTALLLEVEL property in the Property Manager.***

InstallScript and InstallScript MSI Projects

The **FeatureSelectItem** function enables you to select or deselect a feature displayed in a feature-selection dialog such as **SdFeatureTree**.

Conditionally Hiding Features



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*

- *Transform*

The procedure for conditionally hiding features at run time depends on the project type that you are using.

Basic MSI, MSI Database, and Transform Projects

Any feature given an Install Level of zero will be hidden (and deselected).



Task ***For example, to hide a feature if the end user running your installation does not have administrative privileges:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature that you want to configure.
3. Click the **Condition** setting and then click the ellipsis button (**...**). The **Feature Condition Builder** dialog box opens.
4. Click the **New Condition** button. InstallShield adds a new condition row to the **Conditions** box.
5. In the **Level** column, type **0**.
6. In the **Condition** column, type **Not AdminUser**.
7. Click **OK**.

After rebuilding your project and running the installation, the feature will not be displayed or installed if the end user does not have administrative privileges.

InstallScript and InstallScript MSI Projects

The **FeatureSetData** function accepts a FEATURE_FIELD_VISIBLE constant that lets you control whether a specific feature is displayed. For example, to hide a feature called HiddenFeature, include the following function call in your script:

```
FeatureSetData (MEDIA,
    "HiddenFeature",
    FEATURE_FIELD_VISIBLE, FALSE,
    "");
```



Note • *Hiding a feature does not automatically deselect it. To deselect the feature so that its data is not installed, call the following:*

```
FeatureSelectItem(MEDIA, "FeatureName", FALSE);
```

Requiring Features to Be Installed



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *MSI Database*
- *Transform*

When you select Yes for a feature's Required setting, the end user cannot deselect it in the CustomSetup dialog (for Basic MSI, MSI Database, and Transform projects), or the **SdFeatureDialog2**, **SdFeatureMult**, or **SdFeatureTree** dialogs (for InstallScript MSI projects). The feature will be installed to the target system.

If the value for the Required setting is No, the feature is installed by default, but the end user can deselect it.

In InstallScript MSI projects, the Required setting is applicable to root-level features during a first-time installation. It is also applicable to subfeatures in an upgrade or patch. This setting is ignored for subfeatures during a first-time InstallScript MSI installation.

Advertising Features



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *MSI Database*
- *Transform*

InstallShield enables you to selectively enable or disable a feature for advertisement. Advertised features are not installed immediately during the installation process. Instead, they are installed when requested. If the feature is assigned, the feature appears to be already installed, although it is not installed until the end user requests it. (Assigned features have their shortcuts installed, and they can be installed from Add or Remove Programs in the Control Panel. However, an assigned feature is advertised until a user requests it.) A published feature does not appear on the target system until it is requested from the installer. (Published features lack any end user–interface elements. They are installed programmatically or through an associated MIME type.)

Use the Advertised setting in the Features view to specify whether advertisement should be allowed. Available options for this setting are:

Table 4-10 • Options that Are Available for the Advertisement Setting

Option	Description
Allow Advertise	End users have the ability to select the advertisement option for this feature in the CustomSetup dialog. Although advertisement is allowed, it is not the default option when the installation is run.
Favor Advertise	The feature is advertised by default. End users can change the advertisement option for a feature in the CustomSetup dialog.
Disallow Advertise	Advertising is not allowed for this feature. End users cannot elect to have the feature advertised in the CustomSetup dialog.
Disable Advertise if Not Supported	Advertisement works only on systems with Internet Explorer 4.01 or later. If the target system does not meet this criterion, advertising is not allowed. If the target system can support advertisement, advertising is allowed.

When you allow feature advertisement, the feature is advertised, regardless of the mode in which the installation is running, as long as no other factors prevent it from being advertised. In the CustomSetup dialog, the end user can control which features are immediately installed and which are available later.

Advertisement usually requires support from the application. For example, your product's spell checker can be advertised. The application interface offers use of the spell checker through a menu command or toolbar button. You must write to check the feature's installation state and install it when the customer clicks the Spell Check command or button.

Configuring a Feature's Install Level Setting



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

For InstallScript MSI projects, this information applies only if no setup types are defined in the project.

The Install Level setting for a feature is compared against the **INSTALLLEVEL** property at run time to determine which features are available for installation. You can use this setting to create specific feature configurations.

Unless the end user deselects features in the CustomSetup dialog, all features with an install level less than or equal to the value of the package's **INSTALLLEVEL** property are installed to the target system.



Note *• You can change the package's* ***INSTALLLEVEL*** *property in the Property Manager.*



Task ***To set a feature's Install Level property:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature that you want to configure.
3. For the **Install Level** setting, type an integer for this feature's install level. The recommended value is **100**.

Setting a Feature's Remote Installation Setting



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Remote Installation setting for a feature determines whether the feature's files are installed on the target system or run from the source medium, such as a CD-ROM or network server. The default value for a new feature is Favor Local, which means that the files in the selected feature are installed on the target system.

Task ***To change the Remote Installation setting so that the feature's files run only from the source medium:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature that you want to configure.
3. In the **Remote Installation** setting, select **Favor Source**.

Tip *• Selecting* ***Favor Parent*** *gives a subfeature the same value as its parent feature.*

Caution *• If any of the feature's components contain a Windows service, select Favor Local for the Remote Installation setting. Although an end user could change the installation state through the CustomSetup dialog, the Windows Installer cannot install a service remotely.*

Using Release Flags with Features

Project *• The following project types include support for release flags:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

Release flags enable you to create different versions of your product without having to create more than one project. There are two steps for filtering features according to release flags: assigning release flags and specifying which flags to include in the release.

Assigning Release Flags to Features

Project *• The following project types include support for release flags:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

Release flags must be set on features that you want to exclude from certain builds. For example, if you have a feature called Add-ons that should be included only in a special edition of your product, you can flag that feature and include it only when needed.

Task ***To add a release flag to a feature:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature that you want to configure.
3. For the **Release Flags** setting, type a string. The string can be any combination of letters or numbers. To have more than one flag on a feature, use a comma to separate the flags.

To learn how to filter features based on release flags, see Release Flags.

Removing Release Flags

Project *• The following project types include support for release flags:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

Task ***To remove a release flag from a feature:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the feature that you want to configure.
3. In the **Release Flags** setting, delete the value.

Reordering Features

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

When your end users install your product, the CustomSetup dialog (in Basic MSI, MSI Database, or Transform installations) or one of the feature selection dialogs (in InstallScript or InstallScript MSI installations) displays the features in the same order that they are displayed in the Features view in InstallShield. You can change the order in which the features are displayed.

Task ***To change the feature order:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Right-click the feature that you want to move and click **Move Up** or **Move Down**. You can also move a feature left or right, thereby making it a subfeature of another feature or a top-level feature.

Tip *• You can also reorder your features by dragging and dropping. Any feature or subfeature can be moved in this way.*

Using the Required Features Setting

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The Required Features setting enables to you specify features that are required by the selected feature. For example, you might have an installation with two features—**ProgramFiles** and **HelpFiles**. If it is necessary that end users install **ProgramFiles** whenever the **HelpFiles** feature is selected, you need to use the Required Features property.

Task ***To arrange this feature requirement:***

1. In the View List under **Organization**, click **Setup Design** or **Features**.
2. Select the **HelpFiles** feature.
3. Click the **Required Features** property and then click the ellipsis button (**...**). The **Required Features** dialog box opens.
4. Select the **ProgramFiles** check box.
5. Click **OK**.

At run time, if the user selects a Custom setup type, feature-selection dialogs such as **SdFeatureTree** will not allow the user to deselect the **ProgramFiles** feature if the **HelpFiles** feature is selected.

Working with Setup Types

Project *• Setup types are available in the following project types:*

- *InstallScript*
- *InstallScript MSI*

To create setup types for a Basic MSI project, use the feature's Install Level setting.

Setup types enable you to provide different versions of your product to your end users. For example, the default setup types are Complete and Custom. The Complete setup type installs all of the files included in your installation. The Custom setup type lets the end user select which features are installed.

Setup types are based on features. You select the features that you would like to associate with each setup type. Then, when an end user selects a certain setup type, only those features that you associated with that setup type are installed.

By default, each project that you create contains predefined setup types. In the Setup Types view, you can add or remove setup types, rename existing setup types, and change which features are associated with each one.

Adding Setup Types



Project • *Setup types are available in the following project types:*

- *InstallScript*
- *InstallScript MSI*

To create setup types for a Basic MSI project, use the feature's Install Level setting.



Task ***To add a setup type:***

1. In the View List under **Organization**, click **Setup Types**.
2. Right-click the **Setup Types** explorer and click **Add**. InstallShield adds a new setup type.
3. Type a new name for the setup type.

Editing Setup Types



Project • *Setup types are available in the following project types:*

- *InstallScript*
- *InstallScript MSI*

To create setup types for a Basic MSI project, use the feature's Install Level setting.



Task ***To change which features are associated with a setup type:***

1. In the View List under **Organization**, click **Setup Types**.
2. In the **Setup Types** explorer, click the setup type that you want to edit. All of the features in your installation project appear in the lower-left pane.
3. Clear the check boxes next to the features that should not be included in the selected setup type. Select the check boxes next to those features that you want included.

Any setup types listed in the **Setup Types** pane are automatically added to your installation project.

Note *• The* ***SetupType2*** *function displays only the standard setup types—Complete and Custom—with fixed description text for both. If you want greater flexibility, call* ***SdSetupTypeEx*** *in your script instead of* ***SetupType2****. (The default code for the* ***OnFirstUIBefore*** *event handler includes a call to the* ***SetupType2*** *function.)*

Tip *• To provide an accelerator key for your setup type, include an ampersand (&) before a letter in the name. For example, the name* **Cu&stom** *becomes the label Cu*<u>*s*</u>*tom, and the end user can select it during installation by pressing the S key.*

Renaming Setup Types

Project *• Setup types are available in the following project types:*

- *InstallScript*
- *InstallScript MSI*

To create setup types for a Basic MSI project, use the feature's Install Level setting.

Task ***To rename a setup type:***

1. In the View List under **Organization**, click **Setup Types**.
2. In the **Setup Types** explorer, right-click the setup type that you want to edit and then click **Rename**.
3. Type a new name for your setup type.

This updates the Display Name setting for the selected setup type. The name is displayed in the SetupTypes dialog at run time.

Deleting Setup Types

Project *• Setup types are available in the following project types:*

- *InstallScript*
- *InstallScript MSI*

To create setup types for a Basic MSI project, use the feature's Install Level setting.

Task ***To delete a setup type:***

1. In the View List under **Organization**, click **Setup Types**.
2. In the **Setup Types** explorer, right-click the setup type that you want to delete and then click **Remove**.

Including Redistributables in Your Installation

InstallShield includes many commonly used third-party redistributables, making it easy to add support for popular technologies such as the .NET Framework to your installation. When you add redistributables to your project, the redistributables, plus all of the associated dependencies, are added to your installation. This simplifies the process of packaging redistributables and helps to facilitate consistency for internal or external use.

The Redistributables view (or in InstallScript and InstallScript Object projects, the Objects view) contains all of the InstallShield objects and third-party merge modules that are included with InstallShield. In Basic MSI and InstallScript MSI projects, this view also contains InstallShield prerequisites that you can add to your installation. In InstallScript projects, you can use the Prerequisites view to add InstallShield prerequisites to your installation.

InstallShield Prerequisites

An InstallShield prerequisite is an installation for a product or technology framework that is required by your product. You can add any of the existing InstallShield prerequisites to your installation projects and configure many of their settings. You can also create your own InstallShield prerequisites, and add them to your projects.

InstallShield includes a base set of InstallShield prerequisites. You can also use the InstallShield Prerequisite Editor in InstallShield to define custom InstallShield prerequisites or to edit settings for any existing InstallShield prerequisites.

InstallShield includes support for two types of InstallShield prerequisites:

- **Setup prerequisite**—The installation for this type of prerequisite runs before your installation runs.
- **Feature prerequisite**—This type of prerequisite is associated with one or more features. It is installed if the feature that contains the prerequisite is installed and if the prerequisite is not already installed on the system. Thus, if a feature has a condition that is not met on the target system, or if the end user chooses not to install the feature, the feature is not installed. As a result, none of its associated feature prerequisites are installed, unless the feature prerequisites are also associated with other features that are installed.



***Project** • Basic MSI projects include support for feature prerequisites.*

InstallShield also includes support for including InstallShield prerequisites as packages in Advanced UI and Suite/Advanced UI projects. For more information, see Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project.

Merge Modules

A merge module (or .msm file) contains all of the logic and files needed to install distinct pieces of functionality. For example, some applications require Microsoft C++ run-time libraries. Instead of having to include the file in a feature and figure out its installation requirements, you can simply attach the Microsoft C++ runtime library merge module to one of your project's features.



***Note** • Many of the merge modules included in the Redistributables view are authored by Microsoft or another third party. InstallShield distributes these modules as a courtesy to assist you in creating your installation project. However, InstallShield cannot modify or fix any problems that may exist within third party-authored modules. You are encouraged to contact the vendor regarding issues with specific third party-authored modules.*

Objects

Like merge modules, objects contain logic and files needed to install distinct pieces of functionality. Some objects, such as the DirectX object included with InstallShield, require customization through a wizard. As soon as you add such an object to your installation, its customization wizard opens. You can either customize your object at the time you add it, or cancel the wizard and customize your object later by right-clicking the object and selecting Change Objects Settings.

Redistributables Gallery

Because the file size of many of the redistributables is so large, some that are available for use in your projects are not added to your computer when you install InstallShield. However, these redistributables are still available for download from the Internet to your computer.

Configurable Merge Modules

A configurable redistributable is a merge module or an object that has at least one row in the **ModuleConfiguration** table that is referenced by at least one row in the **ModuleSubstitution** table. This enables you to change a value in the redistributable. When you select a configurable module in the Redistributables view, the Merge Module Configurable Values dialog box is displayed to enable you to configure the module at the time you add it. To customize the merge module later, right-click it and select Configure merge module.

Shipping Redistributable Files

InstallShield provides third-party redistributables that you can incorporate into your installation projects. If you include redistributable technology in your projects—for example, Crystal Reports—that redistributable must be licensed from the vendor. You cannot legally redistribute these technologies without the appropriate licensing. For details, consult the vendor's documentation.

When you build releases in an InstallShield project, InstallShield includes various InstallShield redistributable files in the build output. You may use these InstallShield redistributable files in the build output according to the InstallShield End-User License Agreement. Most of these files are installed in the *`InstallShield Program Files Folder`*\Redist folder and included in builds as needed. Following is a list of the InstallShield redistributable files.

- _isres_LanguageID.dll
- AppxStub.exe
- ClrSuitePSHelper.dll
- ClrWrap.dll
- CommonHelper.dll
- corecomp.ini
- default.pal
- DLLWrap.dll
- dotnetfx.exe
- DotNetInstaller.exe
- EulaScrollWatcher.dll

- FileBrowse.dll
- IISHelper.dll
- IISRT.dll
- InstallShield.ClrHelper.dll
- InstallShield.Interop.Msi.dll
- ISBEW64.exe
- ISChain.exe
- ISChainPackages.dll
- ISComSrv.dll
- ISExpHlp.dll
- isexternalui.dll
- IsLockPermissions.dll
- ISNetAPI.dll
- ISNetApiRT.dll
- ISNetworkShares.dll
- ISRegSvr.dll
- Isrt.dll
- ISScheduledTasks.dll
- IsSchRpl.dll
- ISSetup.dll
- ISSQLSrv.dll
- IsWebDeploy.dll
- ISWindowsFeaturesAction.dll
- ISWindowsFeaturesAction64.dll
- ISXmlCfg.dll
- Layout.bin
- PowerShellWrap.dll
- PrqLaunch.dll
- QuickPatchHelper.dll
- SerialNumCAHelper.dll
- SetAllUsers.dll
- Setup.exe
- Setup.ini

- setup.inx
- setup.isn
- setup.ocx
- setup.skin
- Setup_UI.dll
- setupPreReq.exe
- SetupSuite.exe
- SetupSuite64.exe
- SFHelper.dll
- SQLRT.dll
- SuiteAppxHelper.exe
- SuiteSQLRT.dll
- XMLRT.dll
- Image files that are installed in the subfolders in the following folder:

  *InstallShield Program Files Folder***Support\Themes**

- Image and icon files that are installed in the following folder, as well in this folder's scale-150 and scale-200 subfolders:

  *InstallShield Program Files Folder***Redist\Language Independent\OS Independent**

Managing the Redistributables Gallery

Because the file size of many of the redistributables is so large, some that are available for use in your projects are not added to your computer when you install InstallShield. However, these redistributables are still available for download from the Internet to your computer.

You can identify the status of a redistributable by its icon. Following is a list of the possible icons in the Redistributables view (or in InstallScript projects, the Objects view or the Prerequisites view) and a description of each:

Table 4-11 • Redistributable Icons

Icon	Project Type	Description
	Basic MSI, InstallScript, InstallScript MSI	This InstallShield prerequisite is installed on your computer.
	Basic MSI, InstallScript, InstallScript MSI	This InstallShield prerequisite is not installed on your computer but it is available for download.

Table 4-11 • Redistributable Icons (cont.)

Icon	Project Type	Description
	Basic MSI, InstallScript, InstallScript MSI	This InstallShield prerequisite is included in your project but its location is not listed in one of the directories that is specified on the Prerequisites tab of the Options dialog box.
	Basic MSI, InstallScript, InstallScript MSI	This merge module is installed on your computer.
	Basic MSI, InstallScript MSI	This merge module is stored in a repository and is available for inclusion in your project. For more information about repositories, see Using a Repository to Share Project Elements.
	Basic MSI, InstallScript, InstallScript MSI	This merge module is not installed on your computer but it is available for download.
	Basic MSI, InstallScript MSI	An old version of this merge module is installed on your computer. A new version is available for download.
	Basic MSI, InstallScript, InstallScript MSI	This object is installed on your computer.
	Basic MSI, InstallScript MSI	This object is not installed on your computer but it is available for download.
	Basic MSI, InstallScript MSI	An old version of this object is installed on your computer. A new version is available for download.



Project • *If you add to your project a redistributable that is not installed on your computer, one or more build errors are generated when you build a release. To eliminate the build errors, either remove the redistributable from your project or download it before rebuilding the release. If a redistributable is not installed on your computer,* ***Needs to be downloaded*** *is specified in the Location column for that redistributable.*

InstallShield does not permit you to add a redistributable in the Objects view to an InstallScript project if it is not installed on your computer.

Working with the Redistributables Gallery

The Redistributables view in Basic MSI and InstallScript MSI projects displays the redistributables gallery, which consists of InstallShield prerequisites, merge modules, and objects, that you can include with your installations. The Prerequisites view in InstallScript projects also displays the gallery of InstallShield prerequisites that you can add to your project.

InstallShield Prerequisites

Many InstallShield prerequisites are available in InstallShield. In addition, the InstallShield Prerequisite Editor in InstallShield enables you to modify these prerequisites and create your own. All InstallShield prerequisite (.prq) files are stored in the following location:

*InstallShield Program Files Folder***SetupPrerequisites**

Merge Modules

Merge modules are available from a variety of sources. Although InstallShield includes many redistributable modules, new versions may be available or other software developers may have released a module that you need. In addition, InstallShield enables you to create your own merge modules and add them to your redistributables gallery.

The source of the merge module files listed in the Redistributables view is the folder or folders specified on the Merge Modules tab of the Options dialog box. To access the Options dialog box, on the Tools menu, click Options.

The following directory is the default location for the modules that come with InstallShield:

*InstallShield Program Files Folder***Modules\i386**

Objects

InstallShield provides many redistributable objects and lets you create your own for inclusion in your installations. Furthermore, you may want to add to your projects the objects that other developers created with InstallShield.

The default location for the objects that come with InstallShield is:

*InstallShield Program Files Folder***Objects**

The objects that are included in the above location are listed in the Redistributables view.

Downloading Redistributables to Your Computer

The procedures for downloading redistributables to your computer differ, depending on what type of project you are using.

For Basic MSI and InstallScript MSI Projects

The Redistributables view enables you to download the latest InstallShield prerequisites, merge modules, and objects from the Flexera Software Web site to your computer. If a redistributable is not installed on your computer, **Needs to be downloaded** is specified in the Location column for that redistributable.



Task ***To download a specific InstallShield prerequisite, merge module, or object:***

1. In the View List under **Application Data**, click **Redistributables**.
2. Right-click the InstallShield prerequisite, merge module, or object that you would like to download and then click **Download Selected Item**.

Task ***To download all of the InstallShield prerequisites, merge modules, and objects that are needed for your installation project:***

1. In the View List under **Application Data**, click **Redistributables**.
2. Right-click any InstallShield prerequisite, merge module, or object and then click **Download All Required Items**.

For InstallScript Projects

The Prerequisites view enables you to download the latest InstallShield prerequisites from the Flexera Software Web site to your computer. If a redistributable is not installed on your computer, **Needs to be downloaded** is specified in the Location column for that redistributable.

Task ***To download a specific InstallShield prerequisite:***

1. In the View List under **Application Data**, click **Prerequisites**.
2. Right-click the InstallShield prerequisite that you would like to download and then click **Download Selected Item**.

Task ***To download all of the InstallShield prerequisites that are needed for your installation project:***

1. In the View List under **Application Data**, click **Prerequisites**.
2. Right-click any InstallShield prerequisite and then click **Download All Required Items**.

The Objects view enables you to download the latest merge modules and objects from the Flexera Software Web site to your computer. If a merge module is not installed on your computer, its icon () indicates that it is not installed. InstallShield does not permit you to add a merge module to your InstallScript project if the merge module is not installed on your computer. If an object is not installed on your computer, it is not listed in the Objects view.

Task ***To download a specific merge module:***

1. In the View List under **Application Data**, click **Objects**.
2. In the **InstallShield Objects/Merge Modules** pane, right-click the merge module that you would like to download and then click **Download Selected Item**.

Task ***To download objects:***

1. In the View List under **Application Data**, click **Objects**.
2. In the **InstallShield Objects/Merge Modules** pane, right-click any object and click **Check Web**. The downloads page at the Flexera Software Web site opens in an Internet browser.

3. Select the object that you would like to download and install. You are prompted during the installation to close InstallShield in order to complete the object installation.

Adding InstallShield Prerequisites to the Redistributables Gallery



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*



Task ***To add an InstallShield prerequisite to the redistributables gallery:***

1. Acquire the new or updated InstallShield prerequisite (.prq) file.
2. Using Windows Explorer, copy the new prerequisite to the following location:

   *`InstallShield Program Files Folder`***\SetupPrerequisites**
3. Close InstallShield if it is currently open.
4. Launch InstallShield.

The modifications that you made are reflected in the Redistributables view (in Basic MSI and InstallScript MSI projects) and in the Prerequisites view (in InstallScript projects).

Removing InstallShield Prerequisites from the Redistributables Gallery



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*



Task ***To remove an InstallShield prerequisite from the redistributables gallery:***

1. Close InstallShield.
2. Using Windows Explorer, locate and delete the InstallShield prerequisite that you want to remove from the gallery. InstallShield prerequisites are located in the following directory:

   *`InstallShield Program Files Folder`***\SetupPrerequisites**
3. Launch InstallShield.

The modifications that you made are reflected in the Redistributables view (in Basic MSI and InstallScript MSI projects) or in the Prerequisites view (in InstallScript projects).

Browsing for Merge Modules



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

If a merge module that you would like to add to your project is not listed in the Redistributables view, you can browse to find it and also add it to your project and this view.



Task ***To browse for a merge module:***

1. In the View List under **Application Data**, click **Redistributables**.
2. Right-click an item and click **Browse for Merge Module**. The **Open** dialog box opens.
3. Browse to the merge module file.
4. Click **OK**.

What Happens When You Browse for a Merge Module

InstallShield does not maintain references to merge modules as explicit paths. Instead, it generates a key for a merge module based on the merge module GUID and the merge module locale. When InstallShield needs to access the merge module, it looks in the folders specified in the Merge Module Locations box for a file that matches that key. The Merge Module Locations box is on the Merge Modules tab of the Options dialog box.

When you browse for a merge module, the path to the folder containing the merge module is added to the list of paths in the Merge Module Locations box. In addition, a GUID:Locale key is added to your installation project based on the selected file.

Impact on Your Installation

If two merge modules in the Merge Module Locations box have the GUID:Locale key, only one is included into your installation, even if they have different file names. Because of the way InstallShield uses the Merge Module Locations box to search, it is not possible to predict which merge module will be included.

Limiting the Number of Directories in the Merge Module Locations Box

If you use a shared merge module gallery, there might be earlier or later versions of a merge module in the gallery than what exists on the target machine. For this reason, it is sometimes prudent to try and limit the number of directories in your Merge Module Locations box.



Task ***To limit the number of directories, do one of the following:***

- Using Windows Explorer, copy the merge module that you want into one of the folders that is already listed in the **Merge Module Locations** box.
- Remove the default folders from the search path so that you are referencing only the shared location.

Adding Merge Modules to the Redistributables Gallery

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Task ***To add a merge module to the redistributables gallery:***

1. Acquire the new or updated merge module.
2. Using Windows Explorer, copy the new module to one of the folders specified on the **Merge Modules** tab of the **Options** dialog box.

   The default location for the modules that come with InstallShield is:

   *`InstallShield Program Files Folder`***\Modules\i386**
3. Close InstallShield if it is currently open.
4. Launch InstallShield.

The modifications that you made are reflected in the Redistributables view (in Basic MSI and InstallScript MSI projects) and in the Objects view (in InstallScript projects).

Removing Merge Modules from the Redistributables Gallery

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Task ***To remove a module from the redistributables gallery:***

1. Close InstallShield.
2. Using Windows Explorer, locate and delete the merge module that you want to remove from the gallery. Ensure that you search each directory specified on the Merge Modules tab of the **Options** dialog box.
3. Launch InstallShield.

The modifications that you made are reflected in the Redistributables view (in Basic MSI and InstallScript MSI projects) and in the Objects view (in InstallScript projects).

Note • *If you delete a merge module that is currently associated with your installation, the message* ***[Merge Module Not Found]*** *is displayed to inform you that the module cannot be included in your installation.*

Registering Objects in InstallScript Projects



Project • *This information applies to InstallScript projects.*

When you register an object from within an InstallScript project, InstallShield adds that object to the redistributables gallery, which makes it available for inclusion in other projects.



Task ***To register an object:***

1. In the View List under **Application Data**, click **Objects**.
2. Right-click an item, point to **Advanced**, and click **Register new object**. The **Project Settings** dialog box opens.
3. On the Language-Independent Object Properties tab, specify a media file (**Data1.hdr** file).
4. Specify any other desired settings, and click **OK**.

Incorporating InstallShield Prerequisites, Merge Modules, and Objects in Projects

InstallShield includes many third-party redistributables that are packaged as InstallShield prerequisites, merge modules, and objects. You can add these built-in redistributables to your installation projects. To learn how, see this section of the documentation.



Tip • *For information on creating your own InstallShield prerequisites, merge modules, and objects, see Designing InstallShield Prerequisites and Other Redistributables.*

Adding InstallShield Prerequisites, Merge Modules, and Objects to Basic MSI and InstallScript MSI Projects



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding redistributables to InstallScript projects, see Adding InstallShield Prerequisites, Merge Modules, and Objects to InstallScript Projects.

Two types of redistributables—merge modules and objects—must be associated with a feature in order to be installed. You can associate a single merge module or object with as many features or subfeatures as needed. If no features exist in your installation project when you attempt to add a merge module or object, the Create a New Feature dialog box opens, enabling you to create a feature. If you do not create a feature, these two types of redistributables cannot be added to your installation project.

When you add an InstallShield prerequisite to an InstallScript MSI project, you cannot associate it with one or more features.

When you add an InstallShield prerequisite to a Basic MSI project, it is not associated with any feature by default, and it is called a *setup prerequisite*, since it is run before your main installation runs. If appropriate, you can associate an InstallShield prerequisite with one more features that are currently in your project.



Task ***To add an InstallShield prerequisite, merge module, or object to a Basic MSI or InstallScript MSI project:***

1. In the View List under **Application Data**, click **Redistributables**.
2. If you want merge modules that have been published to a repository to be displayed, right-click a redistributable and ensure that **Show Merge Modules in Repository** is enabled.
3. Select the check box in front of the redistributable that you want to add. If you select an object, the associated wizard opens to guide you through the customization process.
4. For a merge module or an object: In the **Conditional Installation** pane, select the check box for each feature that should contain this redistributable.

   If you are working with a Basic MSI project and you want to associate a prerequisite with a feature: In the **Conditional Installation** pane, select the check box for each feature that should contain this prerequisite. If you do not want to associate the prerequisite with a feature, leave the **Install before feature selection** check box selected. This check box is selected by default when you add an InstallShield prerequisite to a Basic MSI project.



Tip *• The right pane in the Redistributables view shows details about the merge module, object, or InstallShield prerequisite that is selected in the list of available redistributables. Review this details pane to find out information such as which files a redistributable installs. You can hide or show the details pane by clicking the Show Details button in this view.*



Note *• If* ***Needs to be downloaded*** *is specified in the Location column for an InstallShield prerequisite that you added to your project, that InstallShield prerequisite is not installed on your computer. You can download the InstallShield prerequisite from the Internet to your computer if you would like to include it in your project. If you build a release without first downloading one or more required InstallShield prerequisites, and if you specify that the InstallShield prerequisites should be extracted from* **Setup.exe** *or copied from the source media (instead of being downloaded from the Web to the end user's computer), one or more build errors may be generated. To eliminate the build errors, remove the InstallShield prerequisite from your project, download it to your computer, or change the InstallShield prerequisite location for the release to the download option; then rebuild the release.*

Obtaining InstallShield Prerequisites and Objects

Note that some of the InstallShield prerequisites and objects are not installed with InstallShield. You may need to download them. For more information, see Obtaining Updates for InstallShield.

Also note that if you have an installation (for example, a **Setup.exe** file or an .msi package) that you want to launch during your installation, you can create your own custom InstallShield prerequisite, and then add that InstallShield prerequisite to your projects as needed. To learn how to create your own InstallShield prerequisite, see Defining InstallShield Prerequisites.

Adding InstallShield Prerequisites, Merge Modules, and Objects to InstallScript Projects

Project *• This information applies to InstallScript projects.*

For information on adding redistributables to Basic MSI and InstallScript MSI projects, see Adding InstallShield Prerequisites, Merge Modules, and Objects to Basic MSI and InstallScript MSI Projects.

Merge modules and objects must be associated with a feature in order to be installed during an InstallScript installation. You can associate a single merge module or object with as many features or subfeatures as needed. If no features exist in your installation project when you attempt to add a merge module or object, the Create a New Feature dialog box opens, enabling you to create a feature. If you do not create a feature, these two types of redistributables cannot be added to your installation project.

When you add an InstallShield prerequisite to an InstallScript project, you cannot associate it with one or more features.

Task

To add an InstallShield prerequisite to an InstallScript project:

1. In the View List under **Application Data**, click **Prerequisites**.
2. Select the check box in front of the InstallShield prerequisite that you want to add.

Tip *• The right pane in the Prerequisites view shows details about the InstallShield prerequisite that is selected in the list of available redistributables. Review this details pane to find out information such as which files a redistributable installs. You can hide or show the details pane by clicking the Show Details button in this view.*

Note *• If* ***Needs to be downloaded*** *is specified in the Location column for an InstallShield prerequisite that you added to your project, that InstallShield prerequisite is not installed on your computer. You can download the InstallShield prerequisite from the Internet to your computer if you would like to include it in your project. If you build a release without first downloading one or more required InstallShield prerequisites, and if you specify that the InstallShield prerequisites should be included with the media (instead of being downloaded from the Web to the end user's computer), one or more build errors may be generated. To eliminate the build errors, remove the InstallShield prerequisite from your project, download it to your computer, or change the InstallShield prerequisite location for the release to the download option; then rebuild the release.*

Task

To add a merge module or object to an InstallScript project:

1. In the View List under **Application Data**, click **Objects**.
2. In the **Features** pane, select the feature to which you want to add an object or merge module.
3. Right-click the object or merge module that you want to add and select **Add to selected feature**. (You can instead drag the object or merge module and drop it on the feature.) For some objects, an associated wizard appears to guide you through the customization process.



Note *• Merge modules that are added to a feature in an InstallScript project appear in the Features pane as subitems of the Merge Module Holder object. This object requires the Windows Installer engine. For more information, see Adding Windows Installer Redistributables to Projects.*



Tip *• To see information about an object or merge module, such as files it installs and other actions it performs, select the object or merge module name. The information is displayed in the pane on the right.*



Caution *• You should not alter another company's merge module.*

Obtaining Objects

Note that some of the objects are not installed with InstallShield. You may need to download them. For more information, see Obtaining Updates for InstallShield.

Removing InstallShield Prerequisites from a Project



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*



Task ***To remove an InstallShield prerequisite from your project:***

1. In the View List under **Application Data**, click **Redistributables** (in Basic MSI and InstallScript MSI projects) or **Prerequisites** (in InstallScript projects).
2. Clear the check box in front of the InstallShield prerequisite that you want to remove.



Tip *• If an InstallShield prerequisite is included in your installation as a dependency of another InstallShield prerequisite but you want to remove that dependency prerequisite from the installation, you must remove the corresponding dependency from the InstallShield prerequisite. For more information, see Removing a Dependency from an InstallShield Prerequisite.*

Removing Merge Modules and Objects from a Project



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*

- *InstallScript MSI*



Task ***To remove a merge module or object from your project:***

- In Basic MSI and InstallScript MSI projects: In the **Redistributables** view, clear the check box in front of the redistributable.
- In InstallScript projects: In the **Objects** view, right-click the redistributable in the **Features** window and click **Delete from project**.

InstallShield moves the merge module or object from your project. In addition, InstallShield automatically removes from the project any dependencies that are associated with that redistributable.

Determining the Files in InstallShield Prerequisites, Merge Modules, and Objects



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

If you need to see a list of files in an InstallShield prerequisite, a merge module, or an object, you can do so from within the Redistributables view in Basic MSI and InstallScript MSI projects. The right pane in this view shows details about the InstallShield prerequisite, merge module, or object that is selected in the list of available redistributables. This details pane provides information such as which files a redistributable installs. You can hide or show the details pane by clicking the Show Details button in this view. Clicking the Show Details button in the Prerequisites view of an InstallScript project lets you see the files in the selected InstallShield prerequisite.

If you are using an InstallScript project and you want to see details about a merge module or object that is listed in the Objects view, select that object; InstallShield displays a list of the redistributable's files, as well as additional information, in the right pane.



Tip *• For another way to see the files contained in a merge module or object, see Knowledge Base article Q106474. This article contains a link to a downloadable Merge Module Dependency Viewer.*

Working with InstallShield Prerequisites that Are Included in Installation Projects



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

An InstallShield prerequisite is an installation for a product or technology framework that is required by your product. Some examples of InstallShield prerequisites that are included with InstallShield are Java Runtime Environment (JRE) and SQL Server Express Edition. You can add any of the existing InstallShield prerequisites to your installation projects and configure many of their settings. You can also create your own InstallShield prerequisites, and add them to your projects.



Project *• Including InstallShield prerequisites in Windows Installer–based projects enables you to chain multiple installations together, bypassing the Windows Installer limitation that permits only one Execute sequence to be run at a time. The* **Setup.exe** *setup launcher serves as a bootstrap application that manages the chaining.*



Note *• Unlike Windows Installer 4.5 chaining, the InstallShield prerequisite installations are not processed as a single transaction; that is, successful installations are not rolled back after failures in later prerequisites. To learn more about Windows Installer 4.5 chaining support, see Configuring Multiple Packages for Installation Using Transaction Processing.*

The Redistributables view is where you add InstallShield prerequisites to Basic MSI and InstallScript MSI projects. The Prerequisites view is where you add InstallShield prerequisites to InstallScript projects.

Setup Prerequisites vs. Feature Prerequisites



Project *• Basic MSI projects include support for feature prerequisites.*

The following project types include support for setup prerequisites but not feature prerequisites:

- *InstallScript*
- *InstallScript MSI*

An InstallShield prerequisite that is run before the main installation's user interface sequence begins is called a *setup prerequisite*. Setup prerequisites are useful for base applications and technology frameworks that must be installed for all configurations of the installed product or that provide functionality that is used during the installation itself. When you add an InstallShield prerequisite to a project, it is the setup prerequisite type of InstallShield prerequisite by default.

Basic MSI projects enable you to associate InstallShield prerequisites with features in your main installation. When an InstallShield prerequisite is associated with one or more features, it is called a *feature prerequisite*. Feature prerequisites are installed after an end user has chosen which features to install; like merge modules, a feature prerequisite is installed only if one or more of the features that contain it are installed. Thus, feature prerequisites are useful for applications or components that are used by only some configurations of the installed product and are not used during the installation itself.

Review the following sections for more information that will help you determine which type of InstallShield prerequisite will best fit your requirements.

Special Considerations for Setup Prerequisites

Following are some tips to consider if you are including one or more setup prerequisites in your project.

.NET Framework Requirements

If your product requires that the .NET Framework be installed on the target system, you may include the .NET Framework redistributable to your project. If the target system does not have the .NET Framework, it is installed during your installation. For more details, see Adding .NET Framework Redistributables to Projects.

If your installation includes the .NET Framework redistributable and a setup prerequisite that requires that the .NET Framework be present on the target machine—for example, if it installs files to the GAC—you can specify that the .NET Framework should be installed before the setup prerequisite is installed. To learn more, see Specifying Parameters for Installing an InstallShield Prerequisite.

Displaying an Error If an End User Launches the .msi Package Instead of the Setup Launcher

If your Basic MSI or InstallScript MSI installation includes a setup prerequisite and end users launch the .msi package for your product directly, rather than launch the **Setup.exe** setup launcher, the setup prerequisite installation will not run. If the prerequisite is not already present on a target system, your product may not work as expected. This scenario may occur if you build an uncompressed release, where the .msi package is not streamed into the **Setup.exe** file.

To prevent this issue from occurring, you may want to add a type 19 custom action to your Basic MSI or InstallScript MSI project. This custom action would evaluate the same conditions that were configured for the prerequisite on the Conditions tab in the InstallShield Prerequisite Editor. The custom action would verify whether the setup prerequisite is still needed; if it is needed, the type 19 error custom action would display an error message and end the installation.

Special Considerations for Feature Prerequisites

Following are some tips to consider if you are including one or more feature prerequisites in a Basic MSI project.

Windows Installer Requirements

If your project includes a prerequisite that installs the Windows Installer, the prerequisite should be a setup prerequisite, not a feature prerequisite. That is, this prerequisite should not be associated with a feature.

.NET Framework Requirements

If your project include a prerequisite that installs the .NET Framework and your installation requires that the .NET Framework be present—for example, if your installation installs files to the GAC—the .NET Framework prerequisite should be a setup prerequisite, not a feature prerequisite. That is, this prerequisite should not be associated with a feature.

Potential Restart Issues for Feature Prerequisites

If you add an InstallShield prerequisite to your project and it may require a restart, it is recommended that you avoid associating the prerequisite with a feature. If a feature prerequisite does trigger a restart, the ReadyToInstall dialog is displayed again after the restart, and the end user will need to click the Install button again to proceed with the rest of the installation.

Calculations for Disk Space Requirements

When the Windows Installer performs the file costing–related actions, it does not automatically include the disk space that is required by any feature prerequisites. Therefore, if the CustomSetup dialog is displayed at run time, the disk space amounts that are listed for various features may be inaccurate, since they will not account for the disk

space that is required by feature prerequisites. In addition, it is possible that a target system may have sufficient disk space for the main installation, but not for the feature prerequisites. In this scenario, the target system may run out of disk space midway through the installation.

To prevent file-costing issues for a feature prerequisite in your project, you may want to add a custom action that evaluates the same conditions that were configured for the feature prerequisite on the Conditions tab in the InstallShield Prerequisite Editor. This custom action would detect whether the feature prerequisite needs to be run if its associated feature is selected to be installed. If the feature prerequisite would need to be run, the custom action would add a temporary row to the ReserveCost table.

Associating an InstallShield Prerequisite with a Feature in a Basic MSI Project



***Project** • Basic MSI projects include support for associating prerequisites with features.*

To learn about the differences between setup prerequisites and feature prerequisites (which are the two types of InstallShield prerequisites), see Setup Prerequisites vs. Feature Prerequisites.

If an InstallShield prerequisite is associated with a feature in a project, it is considered to be a *feature prerequisite*. If it is not associated with a feature, it is considered to be a *setup prerequisite*.

Whenever you add an InstallShield prerequisite to an installation project, it is automatically added as a setup prerequisite by default. You can make it a feature prerequisite by associating it with one or more features that already exist in your project.



Task ***To associate an InstallShield prerequisite with a feature:***

1. In the View List under **Application Data**, click **Redistributables**.
2. In the list of redistributables, select the InstallShield prerequisite that you want to associate with a feature.

   ***Note** • The InstallShield prerequisite's check box must already be selected; this indicates that it is being included in your project. For more information, see Adding InstallShield Prerequisites, Merge Modules, and Objects to Basic MSI and InstallScript MSI Projects.*
3. In the **Conditional Installation** pane, select the check box of each feature to which you want to add this InstallShield prerequisite.

If you want to associate the prerequisite with a new feature, you must first create it. To learn how to create a new feature, see Creating Features.

If you associate a prerequisite with all of the features in your project and then you later add a new feature, InstallShield does not automatically associate the feature prerequisite with the new feature.



***Note** • Feature prerequisites have some limitations that setup prerequisites do not have. For more information, see Setup Prerequisites vs. Feature Prerequisites.*

Disassociating an InstallShield Prerequisite from a Feature in a Basic MSI Project



Project • *Basic MSI projects include support for associating prerequisites with features.*

To learn about the differences between setup prerequisites and feature prerequisites (which are the two types of InstallShield prerequisites), see Setup Prerequisites vs. Feature Prerequisites.

If an InstallShield prerequisite is associated with a feature in a project, it is considered to be a *feature prerequisite*. If it is not associated with a feature, it is considered to be a *setup prerequisite*.



Task ***To remove an InstallShield prerequisite from a feature:***

1. In the View List under **Application Data**, click **Redistributables**.
2. In the list of redistributables, select the InstallShield prerequisite that you want to disassociate from a feature.
3. In the **Conditional Installation** pane, select the **Install before feature selection** check box. This check box is selected by default when you add an InstallShield prerequisite to a Basic MSI project.



Note • *Setup prerequisites have some advantages over feature prerequisites. For more information, see Setup Prerequisites vs. Feature Prerequisites.*

Specifying the Installation Order of InstallShield Prerequisites



Project • *Basic MSI projects include support for feature prerequisites.*

The following project types include support for setup prerequisites but not feature prerequisites:

- *InstallScript*
- *InstallScript MSI*

To learn about the differences between setup prerequisites and feature prerequisites (which are the two types of InstallShield prerequisites), see Setup Prerequisites vs. Feature Prerequisites.

The Redistributables view is where you specify the order in which InstallShield prerequisites should be installed if you include more than one in a Basic MSI project or an InstallScript MSI project. The Prerequisites view is where you specify the order in which InstallShield prerequisites should be installed if you include more than one in an InstallScript project.



Task ***To specify the order in which the InstallShield prerequisites should be installed on the target machine:***

1. In the View List under **Application Data**, click **Redistributables** (in a Basic MSI or InstallScript MSI project) or **Prerequisites** (in an InstallScript project).
2. Add the necessary InstallShield prerequisites to your project if you have not already done so.
3. Right-click any redistributable and click **Set InstallShield Prerequisite Order**. The **InstallShield Prerequisite Installation Order** dialog box opens.

4. Select a prerequisite in the list and then click the up or down arrow to move it up or down in the order for installation.



Project *• Note that when you are specifying the order in a Basic MSI project, InstallShield does not distinguish between setup prerequisites and feature prerequisites. Thus, if your project contains a mix of setup prerequisites and feature prerequisites, they are all listed in one combined list on the InstallShield Prerequisite Installation Order dialog box. At run time, before the main installation launches, the* **Setup.exe** *setup launcher evaluates only the setup prerequisites and—if appropriate—installs them in the order that you specified on the InstallShield Prerequisite Installation Order dialog box. Then later during the installation, the Windows Installer engine evaluates only the feature prerequisites and—if appropriate—installs them in the order that you specified.*



Tip *• If the Windows Installer engine, the .NET Framework, or both must be installed before an InstallShield prerequisite is installed, you can open the InstallShield prerequisite in the InstallShield Prerequisite Editor and specify this requirement. For more information, see Specifying Parameters for Installing an InstallShield Prerequisite.*

Configuring a Release that Includes InstallShield Prerequisites



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

When you package a Basic MSI or InstallScript MSI installation that includes InstallShield prerequisites, you can use any one of the following methods for supplying the InstallShield prerequisite files to end users:

- Store the InstallShield prerequisite files on the source media.
- Compress the InstallShield prerequisite files into **Setup.exe**, to be extracted at run time, as needed.
- If necessary, your installation can download the InstallShield prerequisite files that are included in your project from the URL that is specified in the InstallShield prerequisite file (.prq) for each prerequisite.

For InstallScript installations that include InstallShield prerequisites, the methods that are available are slightly different:

- Store the InstallShield prerequisite files on the source media or in **Setup.exe**, depending on how you configure the settings for the release.
- If needed, your installation can download the InstallShield prerequisite files included in your project from the URL specified in the InstallShield prerequisite (.prq) file for each prerequisite.

You can specify different methods for each InstallShield prerequisite in your project. To learn more, see Specifying a Run-Time Location for a Specific InstallShield Prerequisite.

You can also override individual methods at the release level if you want all of the InstallShield prerequisites in a release to be available through the same method. For more information, see Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level.

You can configure an InstallShield prerequisite so that it is installed either before or after any installation of the Windows Installer engine and the .NET Framework. For more information, see Specifying Parameters for Installing an InstallShield Prerequisite.

Specifying the Directories that Contain InstallShield Prerequisites



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The default location for InstallShield prerequisite files (.prq) is:

*`InstallShield Program Files Folder`***\SetupPrerequisites**

InstallShield lets you specify additional or alternative locations on your local machine, or on a network. This flexibility enables you to store InstallShield prerequisites in source code control and to share a common set of InstallShield prerequisites with other team members.

InstallShield offers several ways for specifying the search paths for InstallShield prerequisite files (.prq):

- If you are editing or building from within InstallShield, use the Prerequisites tab on the Options dialog box—which is displayed when you click Options on the Tools menu—to specify a comma-delimited list of machine-wide folders and current-user folders.

  InstallShield saves the paths that you specify on the Prerequisites tab in the registry on your machine. The paths that you specify for the current user are stored in the following location:

  HKEY_CURRENT_USER\Software\InstallShield*`Version`*\Professional\Project Settings\PrerequisiteSearchPath

  The paths that you specify for all users are stored in the following location:

  HKEY_LOCAL_MACHINE\Software\InstallShield*`Version`*\Professional\PrerequisiteSearchPath

  The Options dialog box is not available if you are using the Standalone Build to build a release; however, if you want, you can manually add the paths to the registry on a machine that has the Standalone Build.

- If you are building from the command line with ISCmdBld.exe, use the `-prqpath` parameter to specify a comma-delimited list of folders.

  If you use an .ini file to specify ISCmdBld.exe parameters, you can use the PrerequisitePath parameter in the [Mode] section of your .ini file to specify a comma-delimited list of folders.

- If you are building through MSBuild or Team Foundation Server (TFS), use the PrerequisitePath parameter on the InstallShield task. This parameter is exposed as the ItemGroup InstallShieldPrerequisitePath when the default targets file is used. To specify multiple paths, use an ordered array of paths.

Instead of using hard-coded paths, you can use path variables in paths, as in the following example:

<ISProductFolder>\SetupPrerequisites,<ISProjectFolder>\MyCustomPrerequisites

The Redistributables view and the Prerequisites view list the names of the InstallShield prerequisites that correspond with the .prq files that are present in the various search paths that are specified on the Prerequisites tab of the Options dialog box. If the same .prq file is in multiple search paths, InstallShield shows only the first instance that it encounters. InstallShield first searches each path that is listed in the per-user setting on the Prerequisites tab. Then, InstallShield checks each path that is listed in the machine-wide setting.

At build time, if your project includes one or more InstallShield prerequisites, InstallShield (or the Standalone Build) searches the specified locations and includes the appropriate InstallShield prerequisites in your release as needed. If the same .prq file is in multiple search paths, InstallShield includes in the build only the first instance that it encounters. It uses the following order to search for .prq files:

1. InstallShield (or the Standalone Build) checks the paths that are specified through the `-prqpath` command-line parameter, the PrerequisitePath .ini file parameter, or the PrerequisitePath parameter on the InstallShield task.
2. InstallShield checks each path that is listed in the per-user setting on the Prerequisites tab. The paths are saved in the following location in the registry.

   HKEY_CURRENT_USER\Software\InstallShield*Version*\Professional\Project Settings\PrerequisiteSearchPath

   Note that the Prerequisites tab is not applicable to the Standalone Build, but the registry path is applicable.
3. InstallShield checks each path that is listed in the machine-wide setting on the Prerequisites tab. The paths are saved in the following location in the registry.

   HKEY_LOCAL_MACHINE\Software\InstallShield*Version*\Professional\PrerequisiteSearchPath

   Note that the Prerequisites tab is not applicable to the Standalone Build, but the registry path is applicable.
4. If no paths are specified in any of the aforementioned locations, InstallShield checks the default location (*`InstallShield (or Standalone Build) Program Files Folder`***\SetupPrerequisites**).



Tip *• If you are using the Standalone Build to build a release that includes InstallShield prerequisites but you do not specify one or more search paths, the Standalone Build searches the default path (`<ISProductFolder>`***\SetupPrerequisites***) for the InstallShield prerequisite files. However, if you specify one or more search paths and do not explicitly include the default path, the Standalone Build does not search the default path.*

Specifying a Run-Time Location for a Specific InstallShield Prerequisite



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

InstallShield enables you to specify a different run-time location for each InstallShield prerequisite in your project.



Task ***To specify a different run-time location for each InstallShield prerequisite in your installation:***

1. In the View List under **Application Data**, click **Redistributables** (in a Basic MSI or InstallScript MSI project) or **Prerequisites** (in an InstallScript project).
2. Select the check box for one of the InstallShield prerequisites that you want to include in your installation.
3. Right-click the InstallShield prerequisite and click **Properties**. The **InstallShield Prerequisites Properties** dialog box opens.
4. In the **Build Location** list, click the appropriate option.

Note that the location that you specify can be overridden at the release level. To avoid overriding the value that you selected for an individual InstallShield prerequisite, the InstallShield Prerequisites Location setting at the release level must be set to Follow Individual Selections. For more information, see Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level.



Tip • *If an InstallShield prerequisite is added to a project as a dependency of another prerequisite, the location for the InstallShield prerequisite dependency follows the location setting of the InstallShield prerequisite that requires it.*

Assigning Release Flags to InstallShield Prerequisites



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

You can set release flags for InstallShield prerequisites that you want to exclude from certain builds. For example, if you have an InstallShield prerequisite that should be included only in a special edition of your product that contains a special add-on that requires the InstallShield prerequisite, you can flag that InstallShield prerequisite and include it only when it is needed.



Task ***To add a release flag to an InstallShield prerequisite that you have added to your installation project:***

1. In the View List under **Application Data**, click **Redistributables**.
2. Right-click the InstallShield prerequisite and click **Properties**. The **InstallShield Prerequisites Properties** dialog box opens.
3. In the **Release Flags** box, type a string. The string can be any combination of letters or numbers. To have more than one flag on a prerequisite, use a comma to separate the flags.

To learn more about filtering InstallShield prerequisites based on release flags, see Release Flags.

Building a Release that Includes InstallShield Prerequisites



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When InstallShield builds a **Setup.exe** file for a project that does not include any prerequisites, it starts with the base **Setup.exe** file that is stored in the following location:

*`InstallShield Program Files Folder`***\redist\Language Independent\i386**

However, when InstallShield builds a **Setup.exe** file for a project that includes prerequisites, the aforementioned files cannot be used as the base because it does not have the capability of including prerequisites. A slightly larger file called **SetupPrereq.exe** is used instead. The base **SetupPrereq.exe** file is located in the same directory as the base **Setup.exe** file. Since different base files—**Setup.exe** and **SetupPrereq.exe**—are used, only installation authors who are actually including prerequisites in their projects incur the additional size overhead in the final, built **Setup.exe** file that is distributed to end users.

Run-Time Behavior for an Installation that Includes InstallShield Prerequisites



Project *• The following project types include support for both setup prerequisites and feature prerequisites:*

- *Basic MSI*

The following project types include support for setup prerequisites but not feature prerequisites:

- *InstallScript*
- *InstallScript MSI*

To learn about the differences between setup prerequisites and feature prerequisites (which are the two types of InstallShield prerequisites), see Setup Prerequisites vs. Feature Prerequisites.

Overview of an Installation that Includes InstallShield Prerequisites

The following procedure explains what typically occurs at run time when an end user launches an installation that includes setup and feature prerequisites. Note that some of the steps apply to only certain project types.

1. The setup launcher (typically called **Setup.exe**) displays the language selection dialog if appropriate.
2. The setup launcher displays the setup prerequisite dialog and launches the setup prerequisite installations if appropriate.
3. The installation displays the installation UI, which may allow the end user to select features or configure items. The installation UI shows a progress dialog.
4. In Basic MSI installations, the setup launcher launches the feature prerequisite installations if appropriate:
    a. The built-in InstallShield custom action ISInstallPrerequisites, which is scheduled between the SetupProgress dialog and the ExecuteAction action, compares the features that were selected for installation against the list in the Windows Installer property **IsPrerequisiteFeatures**. If there are no matches, no feature prerequisites are installed.

b. The ISInstallPrerequisites action attempts to find and launch the setup launcher, and it provides the list of features that are being installed. The path to the setup launcher is identified by the Windows Installer properties **SETUPEXEDIR** and **SETUPEXENAME**:

[SETUPEXEDIR]\[SETUPEXENAME]

If ISInstallPrerequisites cannot find the setup launcher in that location, it searches elsewhere. For a first-time installation, ISInstallPrerequisites checks **SourceDir**. For maintenance mode, ISInstallPrerequisites checks paths that are related to the installation source path.

If ISInstallPrerequisites still cannot find the setup launcher, or if it finds multiple .exe files, the installation prompts the end user to browse to the setup launcher file. If the end user identifies the file, the installation continues. Otherwise, the installation ends.

c. The setup launcher evaluates the list of features to select which feature prerequisites to install, and it launches their installations as appropriate.

5. The installation finishes making changes on the target system according to the end user's selections.

6. The installation switches from the progress dialog to the SetupCompleteSuccess dialog.

The User Interface for an Installation that Includes InstallShield Prerequisites

If a target system needs one or more setup prerequisites to be installed, the setup launcher typically displays the setup prerequisite dialog before the main installation starts. This setup prerequisite lists all of the nonhidden setup prerequisites that are missing from the target system. When an end user clicks the Install button on this dialog, the setup launcher launches the necessary setup prerequisite installations. If one or more of the setup prerequisites is marked as requiring administrative privileges and the installation is run on a system on which User Account Control (UAC) is enabled, the Install button on this dialog has the shield icon to alert the end user that elevated privileges are required.

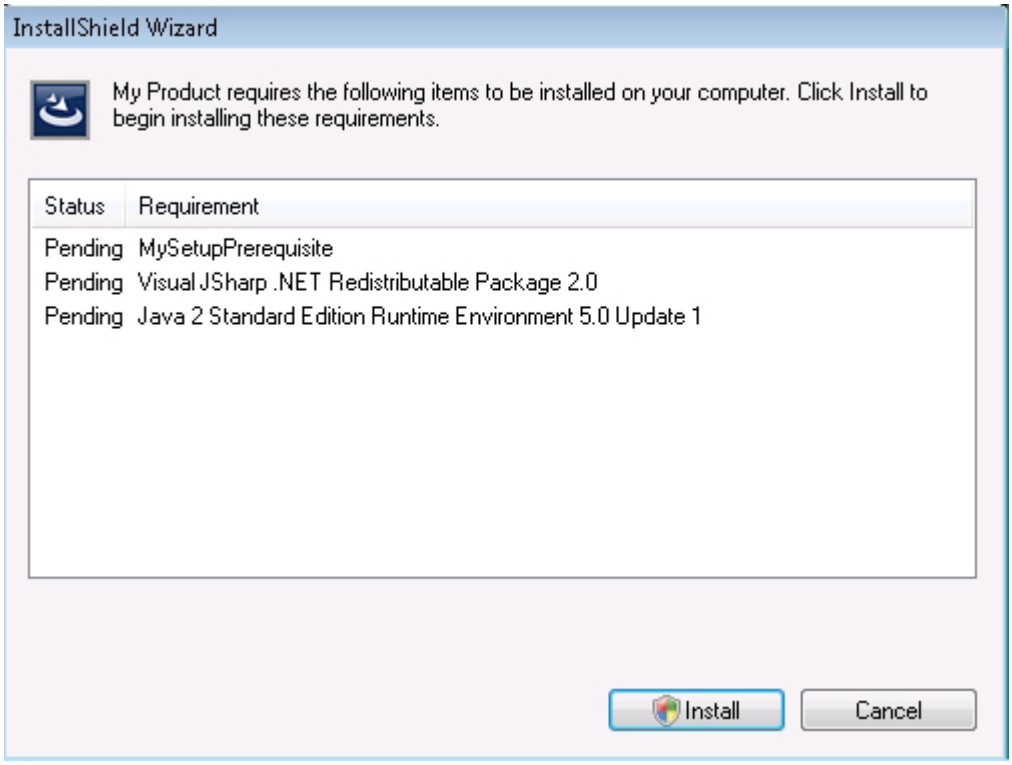


Figure 4-2: Sample Setup Prerequisite Dialog that Shows the List of Setup Prerequisites that Need to Be Installed

If a setup prerequisite is configured to be hidden, it is not listed in the setup prerequisite dialog, but it is still installed. If all of the setup prerequisites in an installation are hidden, the installation displays the setup launcher's standard initialization dialog instead of the setup prerequisite dialog.

Tip • *For instructions on how to show or hide the setup prerequisite in the setup prerequisite dialog, see Specifying Whether to Include the Name of a Prerequisite in the List of Setup Prerequisites to Be Installed on the Target System.*

If the file that a setup prerequisite installation launches is an .msi package and the prerequisite is marked to show progress, the user interface shows a status bar, along with installation progress messages from Windows Installer, while the prerequisite is being installed. This is applicable only if the main installation is a Basic MSI or InstallScript MSI installation. For more details, see Specifying Whether to Show the Progress of an InstallShield Prerequisite Installation at Run Time.

If a setup prerequisite is configured to be optionally installed by the end user, the setup launcher displays a message box that enables end users to choose whether to install the setup prerequisite. For more information, see Allowing End Users to Choose Whether to Install an InstallShield Prerequisite.

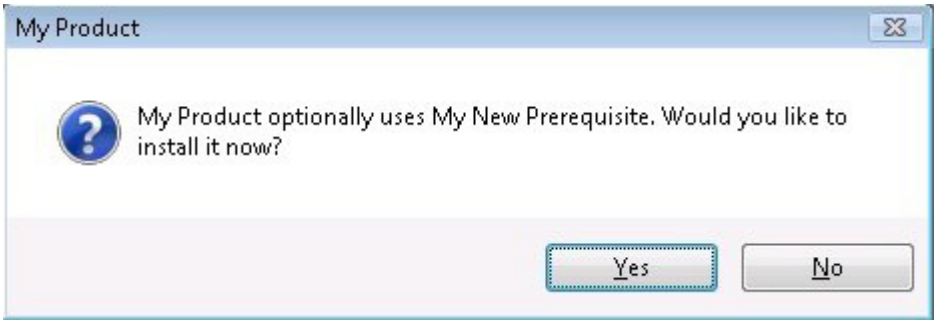


Figure 4-3: Message Box for an Optional Prerequisite

If an installation includes feature prerequisites, the setup launcher does not list them in any prerequisite dialog. However, the user interface does show progress messages if appropriate. In addition, the setup launcher displays the optional prerequisite message box if the feature prerequisite is marked as optional.

Silent Scenarios—Suppressed User Interface in Basic MSI Installations

The installation can install setup prerequisites and feature prerequisites even if the installation is run silently. That is, InstallShield prerequisites are supported in any of the following scenarios:

- **Silent setup launcher and visible .msi package**—The user interface for the setup launcher is suppressed, but the user interface for the .msi package is visible. For example, the end user might use the following command-line statement:

  ```
  Setup.exe /s
  ```

  In this scenario, the language selection dialog and the setup prerequisite dialog are not displayed.

- **Visible setup launcher and silent .msi package**—The user interface for the setup launcher is displayed, but the user interface for the .msi package is suppressed. For example, the end user might use the following command-line statement:

  ```
  Setup.exe /v"/qn"
  ```

  In this scenario, the feature selection dialog and all of the other dialogs of the main installation are not displayed. However, the end user can set Windows Installer properties such as **ADDLOCAL**, **ADDSOURCE**, **ADDDEFAULT**, and **ADVERTISE** from the command line to indicate which features should be installed.

- **Silent setup launcher and silent .msi package**—The user interface for the setup launcher and the .msi package are suppressed. For example, the end user might use the following command-line statement:

  ```
  Setup.exe /s /v"/qn"
  ```

  In this scenario, all of the setup launcher and .msi package dialogs are suppressed.

If the UI sequence of the main installation's .msi package is skipped, the setup launcher evaluates Windows Installer properties such as **ADDLOCAL**, **ADDSOURCE**, **ADDDEFAULT**, and **ADVERTISE** to determine if any feature prerequisites should be installed, and it installs feature prerequisites accordingly.

Silent Scenarios—Suppressed User Interface in InstallScript and InstallScript MSI Installations

An InstallScript or InstallScript MSI installation can install setup prerequisites even if the installation is run silently. That is, InstallShield prerequisites are supported if the end user launches the installation by entering the following command-line statement:

```
Setup.exe /s
```

Note that a response file (Setup.iss) is required. To learn more, see Creating the Response File.

The language selection dialog and the setup prerequisite dialog are not displayed.

UAC Prompts

Depending on how it is configured, an installation that includes InstallShield prerequisites may prompt for elevated privileges on Windows Vista and later systems at several different points during the installation:

1. When the end user launches the **Setup.exe** file
2. When the **Setup.exe** file launches a setup prerequisite that requires elevated privileges
3. When the **Setup.exe** file launches a feature prerequisite that requires elevated privileges
4. When the Windows Installer begins the Execute sequence of the .msi package

For more information, see Minimizing the Number of User Account Control Prompts During Installation.

Changing the Behavior of InstallShield Prerequisites

The InstallShield Prerequisite Editor enables you to modify an InstallShield prerequisite if you want to change its installation's behavior. For example, you can specify whether end users may skip the prerequisite installation, whether the prerequisite requires administrative privileges, and how the prerequisite installation should proceed if it appears that the target machine does or does not need to be restarted. To learn more, see Specifying Installation Behavior for an InstallShield Prerequisite.

You can configure an InstallShield prerequisite so that it is installed either before or after any installation of the Windows Installer engine and the .NET Framework. For more information, see Specifying Parameters for Installing an InstallShield Prerequisite.

Uninstalling an Application Whose Installation Included InstallShield Prerequisites



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Your installation may consist of your application plus one or more InstallShield prerequisites. If end users uninstall your application through Add or Remove Programs in the Control Panel, the InstallShield prerequisites are still installed on their machines. If an InstallShield prerequisite installation added an entry to Add or Remove Programs, an end user would be able to remove that InstallShield prerequisite through Add or Remove Programs.

Working with Merge Modules that Are Included in Installation Projects

This section of the documentation offers guidance for working with merge modules from within installation projects.

For information on creating or modifying your own merge modules, see Designing Merge Modules.

Specifying the Directories that Contain Merge Modules



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

InstallShield lets you specify the locations on your local machine, or on a network, where you are storing merge modules (.msm files). This flexibility enables you to store merge modules in source code control and to share a common set of merge modules with other team members.

InstallShield offers several ways for specifying the search paths for merge modules:

- If you are editing or building from within InstallShield, use the Merge Modules tab on the Options dialog box—which is displayed when you click Options on the Tools menu—to specify a comma-delimited list of machine-wide folders and current-user folders.

  InstallShield saves the paths that you specify on the Merge Modules tab in the registry on your machine. The paths that you specify for the current user are stored in the following location:

  HKEY_CURRENT_USER\Software\InstallShield*Version*\Professional\Project Settings\MMSearchPath

  The paths that you specify for all users are stored in the following location:

  HKEY_LOCAL_MACHINE\Software\InstallShield*Version*\Professional\MMSearchPath

  The Options dialog box is not available if you are using the Standalone Build to build a release; however, if you want, you can manually add the paths to the registry on a machine that has the Standalone Build.

- If you are building from the command line with ISCmdBld.exe, use the `-o` parameter to specify a comma-delimited list of folders.

  If you use an .ini file to specify ISCmdBld.exe parameters, you can use the MergeModulePath parameter in the [Mode] section of your .ini file to specify a comma-delimited list of folders.

- If you are building through MSBuild or Team Foundation Server (TFS), use the MergeModulePath parameter on the InstallShield task. This parameter is exposed as the ItemGroup InstallShieldMergeModulePath when the default targets file is used. To specify multiple paths, use an ordered array of paths.

Instead of using hard-coded paths, you can use path variables in paths, as in the following example:

<ISProductFolder>\MergeModules,<ISProjectFolder>\MyCustomMergeModules

The Redistributables view and the Objects view list the names of the merge modules that correspond with the merge modules that are present in the various search paths that are specified on the Merge Modules tab of the Options dialog box. If the same merge module is in multiple search paths, InstallShield shows only the first instance that it encounters. InstallShield first searches each path that is listed in the per-user setting on the Merge Modules tab. Then, InstallShield checks each path that is listed in the machine-wide setting.

At build time, if your project includes one or more merge modules, InstallShield (or the Standalone Build) searches the specified locations and includes the appropriate merge modules in your release as needed. If the same merge module is in multiple search paths, InstallShield includes in the build only the first instance that it encounters. It uses the following order to search for merge modules:

1. InstallShield checks each path that is listed in the per-user setting on the Merge Modules tab. The paths are saved in the following location in the registry.

   HKEY_CURRENT_USER\Software\InstallShield*Version*\Professional\Project Settings\MMSearchPath

   Note that the Merge Module tab is not applicable to the Standalone Build, but the registry path is applicable.

2. InstallShield checks each path that is listed in the machine-wide setting on the Merge Modules tab. The paths are saved in the following location in the registry.

   HKEY_LOCAL_MACHINE\Software\InstallShield*Version*\Professional\MMSearchPath

   Note that the Merge Module tab is not applicable to the Standalone Build, but the registry path is applicable.

3. InstallShield (or the Standalone Build) checks the paths that are specified through the `-o` command-line parameter, the MergeModulePath .ini file parameter, or the MergeModulePath parameter on the InstallShield task.

4. If no paths are specified in any of the aforementioned locations, InstallShield checks the following default directories, in order:

   a. *InstallShield (or Standalone Build) Program Files Folder*\System

   b. *InstallShield (or Standalone Build) Program Files Folder*\Modules\i386

   c. *InstallShield (or Standalone Build) Program Files Folder*\Objects

   d. *InstallShield (or Standalone Build) Program Files Folder*\Modules\i386\Japanese

   e. *InstallShield (or Standalone Build) Program Files Folder*\Modules\i386\German

   f. *Program Files Folder*\Common Files\Merge Modules



***Tip** • If you are using the Standalone Build to build a release that includes merge modules but you do not specify one or more search paths through any of the user-defined methods (that is, in the registry, through the command line or the .ini file, or through the InstallShield task), the Standalone Build searches the aforementioned default directories for the merge modules. However, if you specify one or more search paths and do not explicitly include the default directories, the Standalone Build does not search the default directories.*

Overriding a Merge Module's Destination

Although you should not alter a third-party merge module, you can override the destination for an InstallShield-created merge module and some third-party merge modules.



***Note** • This procedure redirects only the **TARGETDIR** directory in the merge module, or directories that derive directly from **TARGETDIR**. If the merge module is configured to send files to a predefined folder (for example, SystemFolder), you cannot override the module's destination.*

Task ***To override a merge module's destination:***

1. In the View List under **Application Data**, click **Redistributables**.
2. Select the check box next to the merge module to add it to your installation.
3. Right-click the module and click **Properties**. The Merge Module Properties dialog box opens.
4. In the **Destination** box, type a destination or select one from the list of predefined destinations.
5. Click **OK**.
6. In the **Conditional Installation** pane, select the feature or features that should contain the merge module.

Troubleshooting Merge Module Issues

Deleting an Associated Merge Module

If you delete a merge module that is currently associated with your installation, the message [Merge Module Not Found] is displayed, thereby making you aware that the module cannot be included in your installation.

Changing the Language Property of a Merge Module

If you change the language property of a user-defined merge module or another company's merge module after it has been added to the redistributables gallery, a build error occurs if you try to associate the changed merge module with your project. This error occurs because InstallShield uses the language identifier—along with the module identifier— to identify the module file.

If the language identifier of a merge module file is changed after the file is added to the redistributables gallery, InstallShield cannot find the file.

To locate the correct merge module file, you can browse to it from within the Redistributables view. To learn how, see Browsing for Merge Modules.

Publishing a Merge Module to a Repository from Within an Installation Project

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

If you have an existing merge module that you would like to reuse in Windows Installer–based installation projects or share with other users, you can publish it to a repository.

Task ***To publish a merge module to a repository when you are working on a Windows Installer–based installation project:***

1. In the View List under **Application Data**, click **Redistributables**.
2. Right-click the merge module and then click **Publish Wizard**. The **Publish Wizard** opens.

3. Complete the panels in the Publish Wizard.

Merge modules are built into an installation at build time. If you make a change to a repository merge module and then republish it to the repository, any project that has the old version of the repository merge module will be updated the next time that a release of the project's installation is rebuilt.

Adding Windows Installer Redistributables to Projects



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Although the Windows Installer is built into most versions of Windows, a Windows Installer–based installation may depend on certain functionality that is available in only the latest versions of Windows Installer. In addition, some InstallScript installations may require a particular version of Windows Installer.

InstallShield enables you to include a redistributable for Windows Installer in your project. The method for adding a Windows Installer redistributable to your project depends on the project type that you are using, as well as the version of Windows Installer that your installation requires.

For a list of the minimum operating system requirements for each version of Windows Installer, see Target System Requirements. For a list of which versions of Windows Installer were released with which versions of Windows, see Released Versions of Windows Installer in the Windows Installer Help Library.

Note that Windows Installer 5 and Windows Installer 4 are not available as redistributables.

Basic MSI and InstallScript MSI Projects

InstallShield gives you the option of including with your installation a Windows Installer redistributable in a self-extracting executable file called **Setup.exe**.

Windows Installer Distribution

By default, InstallShield creates a **Setup.exe** setup launcher along with your installation package. The setup launcher is required if you want your installation to install the Windows Installer engine.

If you want to include the Windows Installer redistributable in a Basic MSI or InstallScript MSI project, do one of the following:

- **For Windows Installer 4.5**—Add one or more Microsoft Windows Installer prerequisites to your project. InstallShield includes several versions that target different versions of Windows. For more information, see Including Microsoft Windows Installer Prerequisites.
- **For Windows Installer 3.1, 3.0, or 2.0**—The Setup.exe tab for a release in the Releases view is where you specify information such as whether you want to use a **Setup.exe** launcher, whether you want to include one of these versions of the Windows Installer redistributable, and which version of Windows Installer you want to include. To learn more, see Setup.exe Tab for a Release.

  As an alternative, you can add one or more Microsoft Windows Installer prerequisites to your project. For more information, see Including Microsoft Windows Installer Prerequisites.

***Tip** • You can also specify setup launcher requirements in the Setup Launcher panel of the Release Wizard.*

Overview of the Installation Process

At run time, **Setup.exe** determines if Windows Installer is already installed on the target system. If the Windows Installer is found on the target system and it meets the minimum version requirement, it launches your installation package. If Windows Installer is not installed, or a more recent version needs to be installed, **Setup.exe** installs Windows Installer and then launches your installation package. Note that the system may need to be restarted for Windows Installer to be updated.

InstallScript Projects

In some cases, you may want to add a Windows Installer redistributable to an InstallScript project. For example, you may be using an InstallScript project to chain together multiple Windows Installer–based installations that require a particular minimum version of Windows Installer. In addition, if you add a merge module to an InstallScript project, that merge module must be added as a subitem of the Merge Module Holder object in the Objects view. This Merge Module Holder object requires the Windows Installer engine, so if target systems may not have the Windows Installer present, you need to add the Windows Installer redistributable to your InstallScript project.

If you want to include the Windows Installer redistributable in an InstallScript project, add one or more Microsoft Windows Installer prerequisites to your project. InstallShield includes several versions that target different versions of Windows. For more information, see Including Microsoft Windows Installer Prerequisites.

Including Microsoft Windows Installer Prerequisites

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

InstallShield includes InstallShield prerequisites for several versions of Windows Installer. You can use the Redistributables view (in Basic MSI and InstallScript MSI projects) or the Prerequisites view (in InstallScript projects) to add these InstallShield prerequisites to your project.

Checking for the Presence of Windows Installer 4.5 During Basic MSI or InstallScript MSI Installations

You may want your installation to check the target system to determine if Windows Installer 4.5 is installed; if it is not installed, the installation would display an error message box and prevent the installation from continuing. This may be helpful under the following conditions:

- Your installation depends on Windows Installer 4.5 features.
- It is possible that the .msi file might be used to install your product, rather than the **Setup.exe** setup launcher (which would install Windows Installer 4.5 if appropriate).

For example, if you deploy your installation as an uncompressed release, end users would be able to launch the .msi file directly rather than the **Setup.exe** file. In addition, if systems administrators want to customize your installation for deploying your product throughout an enterprise environment, they might create an administrative installation version and deploy that.

To determine whether Windows Installer 4.5 is installed and display an error message if it is not, create an error custom action (a type 19 custom action). The following procedure explains how.



Task ***To determine whether Windows Installer 4.5 is installed and display an error message if it is not:***

1. In the **Custom Actions and Sequences** view, right-click the **Custom Actions** explorer and click **New Error**.
2. Select the new custom action.
3. In the **Error Message** setting, type the error message text that should be displayed when an end user launches your installation without Windows Installer 4.5 being already installed.
4. In the **Install UI Sequence** and **Install Exec Sequence** settings, select an option that would schedule the error message somewhere before the LaunchConditions action. For example, a common sequence for this error action is **<First Action>**.
5. In the **Install UI Condition** and **Install Exec Condition** settings, type the following condition:

   VersionMsi < "4.05"

Locked File/Reboot Issues

When the InstallShield prerequisite installs the Windows Installer 4.5 engine, the existing engine files are often locked. If your installation needs to use the updated engine before the target system is rebooted, be aware of the following behavior.

On Windows XP and Windows Server 2003 systems, the Windows Installer 4.5 engine files are updated immediately and the requirement for a reboot is noted and delayed until the end of the main installation (or any reboot that precedes it). The main installation uses the updated Windows Installer 4.5 engine, and it requests a reboot at the end of the installation.

On Windows Vista and Windows Server 2008 systems, the Windows Installer 4.5 engine files are not updated until after a reboot, so a reboot is requested immediately. If the end user allows the machine to be rebooted, the installation continues after the reboot. If the end user does not allow the machine to be rebooted, the installation ends. While the reboot is pending, the Windows Installer 4.5 engine appears as not installed, and subsequent attempts to run the prerequisite would result in a failure and would not allow installation to continue. Rebooting the machine completes the installation of Windows Installer 4.5 and allows the installation to continue as planned.

Adding .NET Framework Redistributables to Projects



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

If your product requires that the .NET Framework be installed on the target system, you can add the .NET Framework redistributable to your project. If the target system does not have the .NET Framework, it is installed during your installation.

You can also include redistributables for .NET Framework language packs in your project. The language packs contain translated text, such as error messages, for languages other than English.

The method for adding the .NET Framework and .NET Framework language packs to your project depends on the project type that you are using, as well as the version of .NET Framework that your product requires.



***Note** • Some .NET Framework versions include earlier .NET Framework versions:*

- *.NET Framework 3.5 includes .NET Framework 3.0 SP1 and .NET Framework 2.0 SP1.*
- *.NET 3.0 Framework SP1 includes .NET Framework 2.0 SP1.*
- *.NET 3.0 Framework RTM includes .NET Framework 2.0 RTM .*

Basic MSI and InstallScript MSI Projects

If you want to include .NET support in a Basic MSI or InstallScript MSI project, do one of the following:

- **For .NET Framework 4.5 Full, 4.5 Web, 4.0 Full, 4.0 Client, 3.5 SP1, 3.5, 3.0 SP1, 3.0, 2.0 SP2, 2.0 SP1, or 2.0 (only x64, IA64) redistributables**—Add the appropriate Microsoft .NET Framework prerequisite.

  For more information, see Including the Microsoft .NET Framework and Microsoft .NET Framework Language Pack Prerequisites.

- **For 32-bit .NET Framework 2.0, 1.1, or 1.0 redistributables**—Configure the .NET settings for the release through the .NET/J# tab in the Releases view. As an alternative, you can select the appropriate options through the Release Wizard.

  If you would prefer to distribute InstallShield prerequisites of these redistributables, you can use the InstallShield Prerequisite Editor to create them. For more information, see Defining InstallShield Prerequisites.

To test whether the .NET Framework is already installed on the target system, you can use the built-in **MsiNetAssemblySupport** property. It is set to the version of a particular .NET DLL (**fusion.dll**) if the .NET Framework is installed, and it is not set if the .NET Framework is not installed.

InstallScript Projects

InstallScript projects support two methods for adding the .NET Framework redistributables to your project:

- Use the Prerequisites view to add one or more .NET Framework prerequisites to your project.
- Use the Objects view to add the Microsoft .NET Framework object to your installation. This object also enables you to add one or more language packs to an InstallScript project. To learn more, see Microsoft .NET Framework Object Wizard.

To determine whether a particular version of the .NET Framework or a language pack is installed, use the Is function and pass the DOTNETFRAMEWORKINSTALLED predefined constant.

Obtaining InstallShield Prerequisites and Objects

Note that some of the InstallShield prerequisites and objects are not installed with InstallShield. You may need to download them. For more information, see Obtaining Updates for InstallShield.

Including the Microsoft .NET Framework and Microsoft .NET Framework Language Pack Prerequisites



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

InstallShield includes InstallShield prerequisites for some versions of the .NET Framework and the .NET Framework language packs. You can include these InstallShield prerequisites in Basic MSI, InstallScript, and InstallScript MSI projects if you want to redistribute these versions of the .NET Framework and the language packs.

Following is a list of the .NET Framework redistributables that are available as InstallShield prerequisites. The associated language pack prerequisites are included if available.

- Microsoft .NET Framework 4.5. (One InstallShield prerequisite is for the full package, and one is for the Web package. The Web package is smaller in size, but it requires an Internet connection on the target system at run time.)
- Microsoft .NET Framework 4 Full, which installs the .NET Framework runtime and associated files that are required to run and develop applications that target the .NET Framework 4. (One InstallShield prerequisite is for the full package, and one is for the Web Download package. The Web Download package is smaller in size, but it requires an Internet connection on the target system at run time.)
- Microsoft .NET Framework 4 Client, which installs the .NET Framework runtime and associated files that are required to run most client applications. (One InstallShield prerequisite is for the full package, and one is for the Web Download package. The Web Download package is smaller in size, but it requires an Internet connection on the target system at run time.)
- Microsoft .NET Framework 3.5 SP1 (One InstallShield prerequisite is for the full package, and one is for the Web Download package. The Web Download package is smaller in size, but it requires an Internet connection on the target system at run time.)
- Microsoft .NET Framework 3.5 (One InstallShield prerequisite is for the full package, and one is for the Web Download package. The Web Download package is smaller in size, but it requires an Internet connection on the target system at run time.)
- Microsoft .NET Framework 3.0 SP1 (This is a Web Download package, which requires an Internet connection on the target system at run time.)
- Microsoft .NET Framework 3.0
- Microsoft .NET Framework 3.0 (x64)
- Microsoft .NET Framework 2.0 SP2
- Microsoft .NET Framework 2.0 SP2 (x64)
- Microsoft .NET Framework 2.0 SP2 (IA64)
- Microsoft .NET Framework 2.0 SP1 (x86)
- Microsoft .NET Framework 2.0 SP1 (x64)
- Microsoft .NET Framework 2.0 SP1 (IA64)

- Microsoft .NET Framework 2.0 (x64)
- Microsoft .NET Framework 2.0 (IA64)



Tip *• For information on other versions of the .NET Framework redistributables, see Adding .NET Framework Redistributables to Projects.*

These InstallShield prerequisite installations are run in silent mode. Therefore, the language in which the .NET Framework installation runs is not an issue.

Since there currently is no way to install Windows Installer 3.x on 64-bit Itanium systems running Windows XP, and since .NET Framework 2.0 and later require Windows Installer 3.x or later, the 64-bit .NET Framework 2.0 prerequisites cannot be installed on 64-bit Itanium systems running Windows XP.

The InstallShield prerequisite installations determine if the corresponding version of the .NET Framework is already installed on the target machine by checking the Install or InstallSuccess value data for a particular HKEY_LOCAL_MACHINE key. For more details, you can open any InstallShield prerequisite in the InstallShield Prerequisite Editor and note the conditions that are set.

Including the MySQL Connector ODBC Prerequisite



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

You can include in your installation an InstallShield prerequisite that installs MySQL Connector/ODBC 3.51. It enables end users to connect to a MySQL database server that uses ODBC. Before you can add this InstallShield prerequisite to your projects, you must download the MySQL Connector/ODBC driver and configure the InstallShield prerequisite on your system.



Task ***To add the MySQL Connector ODBC 3.51 prerequisite to your system so that you can add it to your projects:***

1. Open Windows Explorer and browse for the InstallShield prerequisite template folder. The default location is:

   C:\Program Files\InstallShield\2016\SetupPrerequisites\Templates

2. Copy the **MySQL Connector ODBC 3.51.prq** file that is in the Templates folder, and paste it in the InstallShield prerequisite folder. The default location is:

   C:\Program Files\InstallShield\2016\SetupPrerequisites

3. Visit http://dev.mysql.com/downloads/connector/odbc/3.51.html and download the MSI installer for the MySQL Connector/ODBC 3.51 driver for Windows.

4. Save the file in the following location:

   *`InstallShield Program Files Folder`***\Objects\MySQL\Redist**

The next time that you launch InstallShield, the MySQL Connector ODBC prerequisite is available in the Redistributables view.

If you want to change the location on your machine where you store the installer for the MySQL Connector/ODBC 3.51 driver, you can do so by opening the **MySQL Connector ODBC 3.51.prq** file in the InstallShield Prerequisite Editor and modifying the settings. To open the InstallShield Prerequisite Editor, click Prerequisite Editor on the Tools menu. For more information, see Specifying Files for an InstallShield Prerequisite.

Including the InstallShield Prerequisite for the Oracle 11g Instant Client



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

If you want to install the Oracle 11g Instant Client before your installation is run, you can include the Oracle 11g Instant Client prerequisite in your project. Before you can add this InstallShield prerequisite to your projects, you must download Oracle Instant Client and create an .msi package on your system. For more information, see Connecting to an Instance of Oracle and Running SQL Scripts.



***Note** • If you want your installation to download the Oracle 11g Instant Client whenever it needs to be installed on a target system, check with Oracle to ensure that it is allowed. If it is allowed, you must host the download on your own Web site and add its download path to the InstallShield prerequisite. Flexera Software does not make this installation available for download.*

Including the DirectX 9.0 Object

DirectX provides supporting API libraries for multimedia applications and hardware, including the latest graphics cards. If your product requires DirectX to be installed on the target system, you can add the DirectX object—either the Windows Installer–based object for Basic MSI and InstallScript MSI projects, or the InstallScript object for InstallScript projects—to your project. If the target system does not have DirectX, it is installed during your installation.

After installation, the DirectX runtime cannot be uninstalled. DirectX is a system component; end users cannot uninstall it without reinstalling the operating system.



***Tip** • The DirectX object is not installed with InstallShield; you need to download it. To include the DirectX object in Basic MSI or InstallScript MSI projects, obtain the MSI object download. To include the DirectX object in InstallScript projects, obtain the InstallScript object download. To learn more, see Obtaining Updates for InstallShield.*

Redistributable Files

The DirectX objects install all DirectX 9.0c core and optional components, including 32-bit- and 64-bit-specific components.

Including the DirectX Object in Basic MSI and InstallScript MSI Projects

When you add the DirectX object to a Basic MSI or InstallScript MSI project, InstallShield launches the DirectX Object Wizard.

You can use the DirectX object in compressed or uncompressed installations; the DirectX Object Wizard lets you specify whether the DirectX files should be in a folder in the Disk1 folder, or they should be streamed into the .msi file:

- If you specify that the files should be in a folder in the Disk1 folder, InstallShield creates a DirectX folder for your installation at build time and places it in the Disk1 folder of your release. InstallShield lists the DirectX folder in the Disk1 area of the Support Files view (in Basic MSI projects) or the Support Files/Billboards view (in InstallScript MSI projects).
- If you specify that the files should not be in the Disk1 folder, InstallShield embeds the files in your installation's .msi file. InstallShield lists these files in the Language Independent area of the Support Files view (in Basic MSI projects) or the Support Files/Billboards view (in InstallScript MSI projects).



Note *• The custom action that launches the DirectX installation is sequenced in the Execute sequence and run in deferred system context so that it can be run with elevated privileges on Windows Vista and later systems.*

Updating the DirectX Object Files for Basic MSI and InstallScript MSI Projects

If you obtain updates for any of the DirectX files and you want to include them in your DirectX object, place them in the appropriate InstallShield Program Files subfolder with the other DirectX files. The location is:

*`InstallShield Program Files Folder`***\Objects\DirectX9c\Redist**

You can also remove files from that folder if your product does not require them and you do not need to include them in your installation. For more information about the DirectX redistributable files, including details about any updates that Microsoft may have released since the current version of InstallShield was released, see the latest DirectX SDK or Microsoft's Web site (http://msdn.microsoft.com/directx).

At build time, InstallShield uses whatever redistributable files are in that DirectX folder to build your release.

Identifying Application Dependencies



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

A file often relies on functions in other files to perform a task. However, you may not be aware of all these other files—known as *dependencies*—when you include your application's files in your installation project. InstallShield offers scanning wizards to assist you in identifying and working with these files. You can access these scanners in the Dependency Scanners view.

Table 4-12 • Available Scanning Wizards

Scanner	Description
Static Scanning Wizard	Looks at portable executable files (for example, .exe, .ocx, .com, .tlb, .hlp, and .chm) in your project and checks for dependencies that they may require.
Dynamic Scanning Wizard	Monitors your system while an executable file is running and checks for .dll or .ocx files that may be required by the executable file.

Any files that are added to a Basic MSI or InstallScript MSI project through one of these scanners are added in accordance with Setup Best Practices.

If you use the Static and Dynamic scanning wizards, you can specify files that you want to be included or excluded automatically any time that you perform a static or dynamic scan through InstallShield. For more information, see Filtering Files in Dependency Scanners.

Static Scanning



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Static Scanning Wizard enables you to scan the files already added to your project for any dependencies that they may require. This wizard scans all portable executable files (.exe, .dll, .ocx, .sys, .com, .drv, .scr, and .cpl files) in your project and checks for dependencies that they may require. The wizard displays a list of the dependencies that it finds, and it lets you specify whether you want to include each one in your project.

The new files added to your project are added to the same feature as the file that depends upon them, thereby ensuring that they are installed when needed.

How the .NET Scan at Build Setting Affects Static Scanning

In Basic MSI, InstallScript MSI, and Merge Module projects, the value that is selected for a component's .NET Scan at Build setting affects how the Static Scanning Wizard scans the files in that component:

Table 4-13 • Determining How the Static Scanning Wizard Scans Files in a Component

Value of a Component's .NET Scan at Build Setting	Description
None	The Static Scanning Wizard does not scan the .NET portion of the component for dependencies or properties. For example, if your component contains **Notepad.exe** and a .NET assembly file, the Static Scanning Wizard scans **Notepad.exe** for dependencies and does not scan the .NET assembly file for dependencies *or* properties.
Properties Only	The Static Scanning Wizard scans the entire component—all files in the component, including .NET assemblies—for properties, but it does not identify dependencies for the .NET portion of the component. For example, if your component contains **Notepad.exe** and a .NET assembly file, the Static Scanning Wizard scans both files for properties, but it does not identify dependencies for the .NET assembly file.
Dependencies and Properties	The Static Scanning Wizard scans the entire component—all files in the component, including .NET assemblies—for dependencies and properties. Any dependencies that are found during the scan are presented in the File Selection panel. If any properties are found for .NET files, InstallShield adds the properties to the .NET Assembly subnode under the Assembly node in the Advanced Settings area of a component.

Dynamic Scanning



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Dynamic Scanning Wizard is an easy-to-use tool that monitors your system while an executable file runs. The wizard displays a list of .dll and .ocx files that may be required by the executable file, and it lets you specify whether you want to include each one in your project.

The executable file you want to scan can already be included to your project, or it can be added by the wizard.

To learn more about this wizard, see Dynamic Scanning Wizard.

Scanning for 64-Bit Dependencies



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

If you use InstallShield on a 64-bit version of Windows Vista or later or a 64-bit version of Windows Server 2008 or later, the Static Scanning Wizard and the Dynamic Scanning Wizard can scan for 64-bit dependencies of the 64-bit files in your project. These wizards can also scan for 32-bit dependencies of the 32-bit files in your project.

Note that if you use InstallShield on a 32-bit version of Windows, these wizards can scan for only 32-bit dependencies of the 32-bit files in your project. If your project includes 64-bit files, you can manually add any dependencies to the project as needed.

Reviewing Dependency Scanner Results



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Static Scanning Wizard and the Dynamic Scanning Wizard scan your project to help you identify other files that your product may require. Each of these wizards displays a list of possible files and merge modules that you may need to add to your project. Review the scan results carefully, and determine whether it is appropriate to add each specified file and merge module to your project.

Both of the scanners works differently to identify dependencies. In some cases, one of the scanners may identify a dependency that the other does not identify. Therefore, you may want to try using both the static and dynamic scanning wizards for a more complete list of potential dependencies. In some scenarios, a dependency scanner may identify as a dependency a file or merge module that is not required by your product. If that occurs, you can exclude that file or merge module, since the wizards let you include or exclude each identified dependency as needed. In addition, InstallShield enables you to specify on a machine-wide basis any files that you want to be included or excluded automatically any time that you perform a static or dynamic scan through InstallShield. For more information, see Filtering Files in Dependency Scanners.

To obtain the best results when you are trying to identify dependencies, it is recommended that you thoroughly test your product and its installation on a clean machine. If your product does not behave as expected, determine whether any dependencies are missing from the machine, and if so, whether they should be included in the installation.

Filtering Files in Dependency Scanners

When you run the Static and Dynamic scanning wizards, you may find that they list as dependencies certain files that you do not want added to your installation. To avoid having these files added every time you run the scanner, you can edit **Filters.xml**. This file enables you to specify any files that you want the scanners to ignore or include.

Filters.xml is in the following location:

*`InstallShield Program Files Folder`***\Support**

The file must remain in this location after you edit it; otherwise, the scanners will not work properly.



Tip • *You can also use the* **Filters.xml** *file to control which registry items should be excluded during COM extraction. For more information, see Filtering Registry Changes for COM Extraction.*

The **Filters.xml** *file also lets you control which IIS settings should be excluded when you are importing an IIS Web site. For more information, see Filtering IIS Data When Importing a Web Site and Its Settings into an InstallShield Project.*

Excluding Files

The <Exclude> element in the **Filters.xml** file is where you add subelements for each of the files that you want the scanners to exclude. Any files that are listed here will not be added to your installation project by the scanners.

By default, the <Exclude> element has subelements for common system files that are present on all Windows–based machines.

Including Files

The <Include> element in the **Filters.xml** file gives you the ability to override individual files that are subelements of the <Exclude> element. Any files that are listed in subelements of the <Include> element will be added to your installation project by the scanners, even if the files are also listed in subelements of the <Exclude> element.



Note • *The following vital operating system files are never recognized by the scanner, even if you add them as subelements of the <Include> element:*

- ***kernel32.dll***
- ***ntdll.dll***
- ***user32.dll***
- ***gdi32.dll***
- ***advapi32.dll***
- ***shell32.dll***
- ***ole32.dll***

Specifying Files in the <Exclude> and <Include> Elements

If you want to list a file under the <Exclude> or <Include> elements, you must add the file as a subelement. Following is a sample of a properly formatted subelement:

```
<File name="myfile.dll" path="[SystemFolder]" We="needthis"/>
```

Table 4-14 • Recognized Attributes for the <File> Subelement

Attribute	Description
name	This attribute must be lowercase. The value of this attribute (for example, **myfile.dll** in the above example) indicates the name of the file that you want to include or exclude.
path	This attribute is optional. The value of this attribute (for example, **[SystemFolder]** in the above example) indicates the path of the file.

Any other attributes are optional and are not recognized by the scanners. You may want to add additional attributes—such as the **We** attribute in the example above and the corresponding **"needthis"** value—to identify why an item is being excluded or included.

Important • *Ensure that your XML code is well formed; if its not well formed, all of the filters fail. In most cases, you can identify improperly formed XML code by opening the* **Filters.xml** *file in Internet Explorer. You should be able to expand and contract the <Filters>, <Include>, and <Exclude> elements; if you cannot, check the code for errors.*

If you add subelements to the <Exclude> or <Include> elements, be sure that you do not place them within the commented-out section, since InstallShield ignores that area of the **Filters.xml** *file.*

The following sample XML code shows the format of the **Filters.xml** file:

```
<Filters>
 <Include>
  <!--Instructions on how to add files to this element.
  -->
  <File name="mfc42.dll" We="needthis"/>
 </Include>
 <Exclude>
  <!--Instructions on how to add files to this element.
  -->
  <Registry key="HKEY_CLASSES_ROOT\Interface\{00020404-0000-0000-C000-000000000046}"/>
  <File name="12520437.cpx" path="[SystemFolder]" wrp="4.0-10.0" />
  <File name="12520850.cpx" path="[SystemFolder]" wrp="4.0-10.0" />
 </Exclude>
</Filters>
```

Registering COM Servers

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

- *MSM Database*

Most applications require certain COM servers in order to operate properly. In order for the operating system to recognize these COM servers, you need to register them.

With traditional COM, COM data is written directly to the registry. Registry-free COM, or Reg-Free COM, offers a simple alternative to the traditional method. With Reg-Free COM, COM data is written to an application manifest file.

Traditional COM Registration



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

InstallShield supports multiple methods for registering a COM server on a target machine. The following methods are the ones used traditionally:

- The COM information can be extracted from the file at build time (or design time) and written to the COM-related tables of your .msi database.
- The COM server's self-registration function can be called at installation time.

The first method listed—extracting the COM information and writing it to the .msi database—is preferred over the self-registration method. The first method writes the COM class information to the **Class**, **ProgID**, and **Registry** tables of the .msi database.

Determining Whether a COM Server Supports Self-Registration

Not all COM servers support self-registration. Although it is likely that you will know which files in your installation are self-registering, there may be times when you do not know. Because you must handle self-registering files differently than non-self-registering files, you need to be able to determine if a file is self-registering.

In most cases, you can determine whether a file is self-registering if you add it to an InstallShield project. InstallShield tests COM server files when they are added to a project to determine if they are self-registering files. If InstallShield determines that a COM server added to your project is self-registering:

- For Windows Installer–based projects: InstallShield sets the COM Extract at Build property of the file's component to Yes (if you have specified on the Preferences tab of the Options dialog box that COM extraction should occur at build time).
- For InstallScript-based projects: InstallShield marks the file as self-registering.

Extracting COM Information from a COM Server



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Note that for MSI Database and MSM Database projects, COM information cannot be extracted at build time; it is extracted at design time.

The recommended method for registering COM servers is to use Windows Installer tables in the .msi database to make the necessary registry entries for your COM server. With this method, Windows Installer is capable of registering files when they are advertised, and it can safely roll back the registration information in case of a failed installation.

Since each component should have only one portable executable file, the assumption is that all of the registration information belongs to a single file, marked as the key file of its component.

InstallShield supports several methods for extracting COM information from a COM server:

- Use the Component Wizard. You can use this wizard to create a new COM server component. With this method, the wizard performs a one-time scan of COM information from the component's key file.
- For a component's COM Extract at Build setting, select Yes. Since InstallShield extracts the COM data every time that you build a release, this method is helpful if your COM server's interfaces change frequently. The COM server must be the key file of its component for this method.
- Extract the information by right-clicking the file in the Files and Folders view and then clicking Extract COM Data for Key File. Use this method if you have already added the COM server file to a component.

As an alternative to having InstallShield extract the COM data, you can manually add COM information to a component in the component's Advanced Settings.

If you do not want InstallShield to extract COM information every time that you build a release, select No for the COM Extract at Build setting of the component. Use this method if you want InstallShield to register the file according to the information that is statically contained in the component's COM Registration advanced setting. This advanced setting stores information about the file's COM classes, ProgIDs, and so on, that were either extracted in the Component Wizard or entered manually in the Advanced Setting area.

When InstallShield extracts COM information from a COM server, the COM class information is written to the **Class**, **ProgId**, and **Registry** tables of the .msi database.

If you are using InstallShield on a 64-bit operating system, InstallShield can extract COM data from a 64-bit COM server. In order to install the data to the correct locations, the component must be marked as 64 bit. To learn more about 64-bit support, see Targeting 64-Bit Operating Systems.

InstallShield employs a special technique to find all COM information on a COM server without actually registering it. Therefore, COM extraction does not affect the system registry. However, some COM servers' registration processes depend on existing registry entry values. Although InstallShield has developed an algorithm to avoid this scenario, it may not be foolproof in some extreme cases.



Caution *• Some applications, like WinRunner, insert hook .dll files into the COM extraction engine. This causes COM extraction to fail and displays the following message: "ISRegSpy detects following module %1 hooked into this process, which causes ISRegSpy to malfunction. You need to shut the application down and restart COM extraction." If you encounter this message, shut down the application and restart COM extraction, as the dialog box instructs.*

Do not select the self-registering property for .exe files that are not self-registering. To self-register an .exe file, you need to launch the .exe file with the /regserver command. However, if the .exe file does not support the command line switch, the .exe file will be launched during extraction at build time.



Note *• The definitions of the COM-related Windows Installer tables prevent a COM server from being placed in more than one feature. If a COM server needs to be placed in more than one feature, the following are two available options:*

- *Make the second feature that requires the COM server a child of the first, and place the file in the parent feature.*
- *Use self-registration on the COM server instead of using the COM-related Windows Installer database tables.*

Filtering Registry Changes for COM Extraction



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

To prevent InstallShield from extracting undesired COM data from a COM server (either at build or design time), you can edit the **Filters.xml** file and specify the registry keys to be excluded. **Filters.xml** is in the following location:

*`InstallShield Program Files Folder`***\Support**

The file must remain in this location after you edit it; otherwise, COM extraction will not work properly.



Tip *• You can also use the* **Filters.xml** *file to control which files should be included or excluded during dependency scanning. For more information, see Filtering Files in Dependency Scanners.*

The **Filters.xml** *file also lets you control which IIS settings should be excluded when you are importing an IIS Web site. For more information, see Filtering IIS Data When Importing a Web Site and Its Settings into an InstallShield Project.*

Excluding Registry Keys from COM Extraction

The <Exclude> element in the **Filters.xml** file is where you add subelements for each of the registry keys that you want the COM extraction process to exclude. Any keys that are listed here will not be uninstalled when your product is uninstalled.

By default, the <Exclude> element has subelements for common system registry keys that are required.

Specifying Registry Keys in the <Exclude> Element

If you want to list a key under the <Exclude> element, you must add the key as a registry subelement.

Following is a sample of a properly formatted registry subelement that blocks changes to an InprocServer32 registry key, all of its values, and all of its subkeys:

```
  <Registry key="HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{00000231-0000-0010-8000-
00AA006D2EA4}\InprocServer32"/>
```

Following is a sample of a properly formatted registry subelement that blocks changes to only the default value of an InprocServer32 registry key:

```
  <Registry key="HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{00000231-0000-0010-8000-
00AA006D2EA4}\InprocServer32" value=""/>
```

Following is a sample of a properly formatted registry subelement that blocks changes to only the ThreadingModel value name for an InprocServer32 registry key:

```
  <Registry key="HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{00000231-0000-0010-8000-
00AA006D2EA4}\InprocServer32" value="ThreadingModel"/>
```

Table 4-15 • Recognized Attributes for the <Registry> Subelement

Attribute	Description
key	This attribute must be lowercase. The value of this attribute (for example, HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{00000231-0000-0010-8000-00AA006D2EA4}\InprocServer32 in the above examples) indicates the name of the registry key that you want to filter.
value	This attribute is optional: • To block changes to an entire registry key, do not include the value attribute. • To block changes to the default value of the specified key, set the value of this attribute to a null value. • To block changes to the name of a value of the specified key, set the value of this attribute to the name of that registry value.

Any other attributes for the <Registry> subelement are optional and are not recognized by the COM extraction process. You may want to add additional attributes to identify why an item is being excluded.



Important • *Ensure that your XML code is well formed; if its not well formed, all of the filters fail. In most cases, you can identify improperly formed XML code by opening the* **Filters.xml** *file in Internet Explorer. You should be able to expand and contract the <Filters>, <Include>, and <Exclude> elements; if you cannot, check the code for errors.*

If you add subelements to the <Exclude> or <Include> elements, be sure that you do not place them within the commented-out section, since InstallShield ignores that area of the **Filters.xml** *file.*

The following sample XML code shows the format of the **Filters.xml** file:

```
<Filters>
 <Include>
  <!--Instructions on how to add files to this element.
```

```
  -->
 </Include>
 <Exclude>
  <!--Instructions on how to add files to this element.
  -->
  <Registry key="HKEY_CLASSES_ROOT\Interface\{00020404-0000-0000-C000-000000000046}"/>
  <File name="12520437.cpx" path="[SystemFolder]" wrp="4.0-10.0" />
  <File name="12520850.cpx" path="[SystemFolder]" wrp="4.0-10.0" />
 </Exclude>
</Filters>
```

Self-Registering COM Servers



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

If you have a COM server that is self-registering, you can call the COM server's self-registration functions at installation time to register the COM server on the target machine.

InstallShield supports different methods for indicating that a COM-related file (.ocx, .dll, .exe, .tlb, or .olb) is self-registering:

- You can specify that a particular file is self-registering. For more information, see File Properties Dialog Box.
- You can specify that a dynamic link is self-registering. For more information, see Dynamic File Link Settings Dialog Box.



Tip *• If the self-registering file is part of a 64-bit component, 64-bit self-registration occurs on the target machine. For more information, see Targeting 64-Bit Operating Systems.*

When you mark a COM server .dll or .ocx file as self-registering, the file's own registration function (**DllRegisterServer**) is called during installation to register it with the target system, and its unregistration function (**DllUnregisterServer**) is called during uninstallation to unregister the file.

Registering a file with **DllRegisterServer** has several limitations:

- COM information registered with **DllRegisterServer** cannot be advertised.
- Because **DllRegisterServer** is .dll code, its effects cannot reliably be rolled back during a failed installation.
- **DllRegisterServer** cannot distinguish between per-user and per-machine COM information.
- Self-registration may require COM servers to be registered in a particular order. InstallShield self-registration (ISSelfReg) enables you to overcome this limitation.
- **DllRegisterServer** generally does not register a file with a relative path, as needed by systems that support side-by-side sharing.

In addition, if you use any type of self-registration in a patch, the patch will not be uninstallable.

When you mark one or more files as self-registering, InstallShield adds data to a table of your .msi database, and adds some custom actions related to self-registration to your installation Execute sequence. For details, see Self-Registration Methods and InstallShield Self-Registration (ISSelfReg).

Per-machine self-registration information is stored in the registry key HKLM\SOFTWARE\Classes\CLSID, and per-user self-registration information is stored in HKCU\SOFTWARE\Classes\CLSID; a merged view of the data is available under HKCR\CLSID.

Self-Registration Methods



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

On the Preferences tab of the Options dialog box, you can select whether you want to use the InstallShield self-registration method (ISSelfReg) or the Windows Installer self-registration method (SelfReg).



Note • *If you use any type of self-registration in a patch, the patch will not be uninstallable.*

Windows Installer Self-Registration

Windows Installer self-registration does the following:

- Registers only .dll and .ocx files.
- Registers files in arbitrary order.
- Can register 64-bit files if the self-registering file is part of a 64-bit component.

InstallShield Self-Registration

InstallShield self-registration does the following:

- Can register .exe, .tlb, .dll, and .ocx files.
- Uses batch mode registration to handle any registration dependencies at run time.
- Requires the inclusion of a self-registration engine that is 70 KB in overhead.
- Can register 64-bit files if the self-registering file is part of a 64-bit component.



Note • *With InstallShield self-registration, you can specify the order in which the files are registered at run time. You can set the order in the ISSelfReg table (Order column) in the Direct Editor.*

InstallShield Self-Registration (ISSelfReg)



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

If you have selected ISSelfReg as the self-registration method for COM servers, and if your project contains any files (or dynamic links) marked as self-registered, InstallShield adds information about those files to the **ISSelfReg** table of your .msi database. You can view and edit the **ISSelfReg** table in the Direct Editor. To learn about the fields in this table, see ISSelfReg Table.

In addition, if your project contains any files (or dynamic links) marked as self-registered, InstallShield adds the following custom actions to the Execute sequence of your installation.

Table 4-16 • Custom Actions Added to Projects that Have Self-Registering COM Servers

Action	Description
ISSelfRegisterCosting	Immediate-execution action that reads the **ISSelfReg** table and determines which files need to be registered or unregistered. A file will be registered when its component is scheduled to be installed, and unregistered when its component is scheduled to be uninstalled.
ISUnSelfRegisterFiles	Deferred-execution custom action that unregisters each file whose component is scheduled to be removed.
ISSelfRegisterFiles	Deferred-execution custom action that registers each file whose component is scheduled to be installed.
ISSelfRegisterFinalize	Displays error information for files that failed to self-register.

Registry-Free COM Registration



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

With Reg-Free COM, COM data is written to an application manifest file that is stored in the application folder. The manifest file is an XML file that contains information about an application and the libraries that are associated with it. Note that the Reg-Free COM manifest file, the executable file, and the COM libraries should all be installed to the same folder on the target machine.

Benefits of Reg-Free COM

Reg-Free COM has several advantages over traditional COM. For example, with Reg-Free COM, the component is defined within the scope of the application itself. Even if other applications that use the same COM component or a different version of it require that it be registered, it will not interfere with this application.

Problems may occur with traditional COM registration if multiple versions of shared libraries exist on a target system. For example, an installation may overwrite a new version of a shared library with an older version, or a new version might not be backwardly compatible with older versions. This may cause applications that require features of a specific version to crash. These types of situations are commonly known as *DLL Hell*. With Reg-Free COM, you can avoid these problems because other applications cannot access your application's COM component.

In addition, Reg-Free COM streamlines the upgrade and uninstallation processes. For an upgrade, simply replace the application folder. For an uninstallation, simply remove that folder.

Limitations of Reg-Free COM

Reg-Free COM is not appropriate for some solutions. Several limitations exist:

- Reg-Free COM works on only Windows XP or later.
- A component is not suitable for Reg-Free COM if it is a system component or part of the operating system. In addition, it is not suitable if it is a data access component such as Microsoft Data Access Components (MDAC). These types of components should not be isolated.
- A COM component can be isolated only once per application. Consider grouping COM components in a single class library as a workaround to this limitation.

Creating and Modifying Reg-Free COM Files for Your Installation



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Use the Reg-Free COM Wizard to create and modify Reg-Free COM manifest files to be included in your installation. Before you use the Reg-Free COM Wizard, you should add the COM libraries (.dll and .ocx files) and the executable file that uses them to your InstallShield project.

The wizard extracts COM information from the libraries that you specify and adds them to the application manifest files. It also creates a new component with the manifest file set as the key file. The manifest component has the same destination and condition as the associated executable-file component.

Sample Manifest File for a Reg-Free COM File



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The contents of the manifest file include the name of the executable file, the file names of the libraries that are associated with the executable files, and the COM information. For the following sample manifest file, the name of the executable file is **Sample.exe**, and the name of the library is **SampleCircle.dll**.

```
<?xml version="1.0" encoding="utf-8"?>
<assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1 assembly.adaptive.xsd"
manifestVersion="1.0" xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns:asmv2="urn:schemas-
microsoft-com:asm.v2" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" xmlns="urn:schemas-microsoft-
com:asm.v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <assemblyIdentity type="win32" name="Sample.exe" version="1.0.0.10"/>
    <file asmv2:size="24576" name="SampleCircle.dll">
        <hash xmlns="urn:schemas-microsoft-com:asm.v2"> ... </hash>
        <typelib flags="HASDISKIMAGE" helpdir="" resourceid="0" tlbid="{6A2304BC-F200-49B8-955E-
0ACDE272B6E0}" version="1.0"/>
        <comClass clsid="{C621F4D3-8463-4B0C-9BEC-84D979C41385}" description="SampleCircle.Server"
progid="SampleCircle.Server" threadingModel="Apartment" tlbid="{6A2304BC-F200-49B8-955E-0ACDE272B6E0}"/
>
    </file>
</assembly>
```

On a Windows XP system, the presence of the manifest file enables the application to use methods from the COM library without the libraries having been registered in the target system's registry. Windows XP uses the COM libraries in the current directory before using any other versions of the libraries that may be registered on the target system.

Including DIMs in a Project



Project *• This information applies to Basic MSI projects.*

A developer installation manifest (DIM) is a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete portion of a product installation. DIM projects enables release engineers to efficiently reuse functional portions of an installation.

Working with DIMs also enables multiple developers to contribute to the development of the installation simultaneously. Each software developer can work on a separate DIM that the release engineer can reference in one or more installation projects.

This section of the documentation explains how to include a DIM project in a Basic MSI installation project. It also offers guidance on how to work with a Basic MSI project that includes one or more DIMs.



Tip *• To learn how to create a DIM project, see Modularizing Installation Projects to Distribute Development Work and Enable Reuse.*

Determining the Appropriate Method for Incorporating DIMs in Installation Projects



Project *• This information applies to Basic MSI projects.*

A developer installation manifest (DIM) is a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete portion of a product installation. Once you have created a DIM, you can add it to a Basic MSI project in one of two ways:

- **By reference**—You can add a reference for a DIM project to your Basic MSI project. With this method, the DIM elements are merged into the Basic MSI project at build time. Each time that you build the Basic MSI installation, InstallShield references the latest version of the DIM project and includes it in the installation that it generates.

  Using DIM references in Basic MSI projects enables multiple software developers to contribute to the development of the installation simultaneously. Each software developer can work on a separate DIM that the release engineer can reference in one or more installation projects.

  If you want to modify any portion of a DIM project, open the DIM project in InstallShield, make the required changes, and then rebuild any Basic MSI projects that reference that DIM.

  This method is the more commonly performed method.

  To learn more, see Referencing a DIM in a Project.

- **By import**—You can import a DIM project into your Basic MSI project. This method is a one-time, irreversible import that merges the DIM data into the Basic MSI project at design time.

  Any further changes that need to be made to the DIM data are made from within the Basic MSI project. Modifying the original DIM project does not modify the corresponding data in the Basic MSI project.

  You may want to import a DIM project to troubleshoot problems with a product or its installation. When you are importing a DIM project, you have the option to back up your Basic MSI project (.ism) before the import; this option enables you to revert back to the base Basic MSI project without the imported DIM project.

  To learn more, see Importing a DIM into a Project.

DIM projects enables developers and release engineers to reuse functional portions of an installation efficiently.

Referencing a DIM in a Project



Project *• This information applies to Basic MSI projects.*

When you add a reference for a DIM project to your Basic MSI project, the DIM elements are merged into the Basic MSI project at build time. Each time that you build the Basic MSI installation, InstallShield references the latest version of the DIM project and includes it in the installation that it generates.

Task ***To add a reference for a DIM to a specific feature in a Basic MSI project:***

1. In the View List under **Organization**, click **Setup Design**.
2. In the **Setup Design** explorer, right-click the feature that you want to contain the DIM reference, and then click **Add DIM Reference**. The **Open** dialog box opens.
3. Browse to the DIM project file (.dim) that you want to reference, and then select that file.
4. Click the **Open** button.

InstallShield adds to the appropriate feature a component node for each component in the DIM project.

Tip *• You can also use the DIM References view to add a reference for a DIM to your Basic MSI project. In this view, right-click the DIMs explorer and then click Add DIM Reference. When you add a reference through this method, the DIM is not associated with any features in the Basic MSI project. To learn how to set up the association, see Associating a DIM Reference with a Feature. Each referenced DIM must be associated with a feature in the Basic MSI project; otherwise, InstallShield does not include it in the Basic MSI installation that it generates at build time.*

Associating a DIM Reference with a Feature

Project *• This information applies to Basic MSI projects.*

When you are including a DIM reference in a Basic MSI project, the DIM reference must be associated with at least one feature; otherwise, InstallShield does not include it in the Basic MSI installation that it generates at build time.

You can associate a DIM reference with one or more features through the DIM References view or the Setup Design view of the Basic MSI project.

Task ***To use the DIM References view of a Basic MSI project to associate a DIM reference with a feature:***

1. In the View List under **Organization**, click **DIM References**.
2. In the **DIMs** explorer, select the DIM that you want to configure.
3. Click the **Features** tab.
4. Select the check box of each feature that you want to contain the selected DIM. Clear the check box of each feature that should not contain the selected DIM.

Task ***To use the Setup Design view of a Basic MSI project to associate a DIM reference with a feature:***

1. In the View List under **Organization**, click **Setup Design**.
2. In the **Setup Design** explorer, right-click the feature that you want to contain the DIM reference, and then click **Associate DIM References**. The **Associate DIM References** dialog box opens; it lists all of the DIM references that are not associated with a feature in the Basic MSI project.

3. Select each DIM that you want to associate with the selected feature, and then click **OK**.

Resolving Build-Time Conflicts Between a Basic MSI Project and a DIM Reference



Project • *This information applies to Basic MSI projects.*

When you build a Basic MSI release that contains a DIM reference, InstallShield merges the DIM data into the Basic MSI release that it generates. If data conflicts exist, InstallShield needs to determine which data takes precedence: the values in the DIM project or those in the Basic MSI project.

InstallShield lets you specify how you want to resolve conflicts between settings in a DIM project with those in a Basic MSI project that contains a reference to that DIM. Following are examples of different types of conflicts that may occur:

- Both projects contain a property that has the same name but different values.
- Both projects contain a string entry that has the same string identifier but different values for a particular language.
- Both projects contain a path variable that contains the same name but different values.
- Both projects contain a component that has the same name but different values for one or more of its settings.

In most cases, conflicts do not occur, since InstallShield automatically includes a DIM project–specific GUID in the names of items such as properties, string entries, path variables, and components when you create these in a DIM project, but not in a Basic MSI project. If you create a property called **MYPROPERTY** in a Basic MSI project, the property is called **MYPROPERTY**. If you create a property called **MYPROPERTY** in a DIM project, InstallShield internally calls the property **MYPROPERTY.DIM_GUID**, where *DIM_GUID* is the identifier that is defined in the DIM GUID setting of the General Information view of the DIM project. Thus, the name of the property in the DIM becomes, for example, **MYPROPERTY.8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC**. For this particular example, you can use the Direct Editor to see the internal name of the property in the **Property** table of the DIM project.



Task ***To indicate how you want InstallShield to resolve conflicts between a Basic MSI project and one of its DIM references:***

1. In the View List under **Organization**, click **DIM References**.
2. In the **DIMs** explorer, select the DIM that you want to configure.
3. Click the **Build Options** tab.
4. For the **Conflict Resolution** setting, select the appropriate value. Available options are:
   - **Use Base Project Value**—If InstallShield encounters a conflict when merging the DIM data into the Basic MSI installation at build time, InstallShield uses the value that is in the Basic MSI project instead of the value that is in the DIM project.
   - **Use DIM Project Value**—If InstallShield encounters a conflict when merging the DIM data into the Basic MSI installation at build time, InstallShield uses the value that is in the DIM project instead of the value that is in the Basic MSI project.

One possible but unlikely scenario for encountering conflicts is if you import a DIM into a Basic MSI project, modify an item in the DIM project, and then try to add a reference to the same DIM in the Basic MSI project. In this scenario, InstallShield overrides the value of the DIM setting or the value of the Basic MSI setting, depending on how you have the Conflict Resolution setting configured, to avoid a conflict.

Viewing Build Instructions for a DIM Reference



Project • *This information applies to Basic MSI projects.*

If you add a DIM reference to a Basic MSI project, it is recommended that you check to see whether the author of the DIM included any build instructions that indicate guidance, tips, or other comments that may help you determine whether additional steps may be required for the specific DIM.



Task ***To view build instructions for a DIM reference in a Basic MSI project:***

1. Open the Basic MSI project that contains the DIM reference.
2. In the View List under **Organization**, click **DIM References**.
3. In the **DIMs** explorer, select the DIM that you want to configure.
4. Click the **Instructions** tab.

On the Instructions tab, InstallShield displays any comments that the author of the DIM project entered when creating or editing it.

Overriding the Destination for a DIM Reference



Project • *This information applies to Basic MSI projects.*

If you create a DIM project, you have the ability to specify the default location on target systems for the files in the DIM project. The location is typically the value of the property **INSTALLDIR** or a subfolder of that folder. When you add a DIM reference to a Basic MSI project, you can choose to use the destination location that the author of the DIM project configured, or you can override the location and specify a different location.



Task ***To view the default destination that is specified for a DIM reference's files and override it if appropriate in a Basic MSI project:***

1. Open the Basic MSI project that contains the DIM reference.
2. In the View List under **Organization**, click **DIM References**.
3. In the **DIMs** explorer, select the DIM that you want to configure.
4. Click the **Build Options** tab. The **Destination** setting shows the value that author of the DIM project configured.
5. To override the default value, specify the appropriate value.

Instead of hard-coding a path, you can enter a directory property as part of the path. To select a directory property, click the ellipsis button (**...**) in this setting. This enables you to select the appropriate directory from a list, or to create a new directory within a predefined directory. Separate further levels of subdirectories with a backslash—for example, **[ProgramFilesFolder]MyApp\Bin**.

Note that it is also possible to override destinations at the feature and component levels. For more information, see Setting the Default Product Destination Folder (INSTALLDIR).

Scheduling Custom Actions and Dialogs from a DIM Reference

Project • *This information applies to Basic MSI projects.*

If you create a DIM project, you have the ability to create custom actions and dialogs. When you add a DIM reference to a Basic MSI project, you can schedule when you want the custom actions in that DIM to be run during the Basic MSI installation. You can also specify when during the Basic MSI installation the dialogs in that DIM should be run.

Task ***To insert custom actions and dialogs from a DIM reference into a Basic MSI project:***

1. Open the Basic MSI project that contains the DIM reference.
2. In the View List under **Organization**, click **DIM References**.
3. In the **DIMs** explorer, select the DIM that you want to configure.
4. Click the **Sequences** tab.
5. In the **Sequences** explorer, right-click the action or dialog that you want your action to follow and click **Insert**. The **Insert Action** dialog box opens, providing a list of all the actions and dialogs that can be added to the sequence.
6. In the list at the top of the dialog box, select the type of action that you want to insert.
7. In the box that shows a list of actions, select the action that you want to insert.
8. Click **OK**.

Opening a Referenced DIM Project from Within an Installation Project

Project • *This information applies to Basic MSI projects.*

If you are modifying a Basic MSI project that contains one or more DIM references and you want to edit the DIM project or see detailed information about it, you can open that project with the click of a link. Note that you must be able to access the DIM project from the machine on which you are modifying the Basic MSI project.

Task ***To open a referenced DIM project from within a Basic MSI project:***

1. In the View List under **Organization**, click **DIM References**.
2. In the **DIMs** explorer, select the DIM that you want to open.
3. On the **General** tab, click the **Launch DIM Project** link.

A new instance of InstallShield launches, and the DIM project opens within InstallShield.

Importing a DIM into a Project

Project • *This information applies to Basic MSI projects.*

When you import a DIM project into a Basic MSI project, InstallShield merges the DIM data into the Basic MSI project. The import is a one-time, irreversible action.

You may want to import a DIM project to troubleshoot problems with a product or its installation. When you are importing a DIM project, you have the option to back up your Basic MSI project (.ism) before the import; this option enables you to revert back to the base Basic MSI project without the imported DIM project.

Task ***To import a DIM into a specific feature in a Basic MSI project:***

1. In the View List under **Organization**, click **Setup Design**.
2. In the **Setup Design** explorer, right-click the feature that you want to contain the DIM, and then click **Import DIM Wizard**. The **Import DIM Wizard** dialog box opens.
3. Complete each panel of the wizard.

Resolving Design-Time Conflicts While Importing a DIM Into a Basic MSI Project

Project • *This information applies to Basic MSI projects.*

When you import a DIM project into an installation project, InstallShield merges the DIM data into the installation project. If data conflicts exist, InstallShield needs to determine which data takes precedence: the values in the DIM project or those in the Basic MSI project.

InstallShield lets you specify how you want to resolve conflicts between settings in the two projects when you are importing. Following are examples of different types of conflicts that may occur:

- Both projects contain a property that has the same name but different values.
- Both projects contain a string entry that has the same string identifier but different values for a particular language.
- Both projects contain a path variable that contains the same name but different values.
- Both projects contain a component that has the same name but different values for one or more of its settings.

If InstallShield encounters a conflict when importing a DIM project, the Resolve Conflict dialog box opens. This dialog box is where you specify how you want to proceed. To learn more, see Resolve Conflict Dialog Box.

In most cases, conflicts do not occur, since InstallShield automatically includes a DIM project–specific GUID in the names of items such as properties, string entries, path variables, and components when you create these in a DIM project, but not in a Basic MSI project. If you create a property called **MYPROPERTY** in a Basic MSI project, the property is called **MYPROPERTY**. If you create a property called **MYPROPERTY** in a DIM project, InstallShield internally calls the property **MYPROPERTY.DIM_GUID**, where *DIM_GUID* is the identifier that is defined in the DIM GUID setting of the General Information view of the DIM project. Thus, the name of the property in the DIM becomes, for example, **MYPROPERTY.8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC**. For this particular example, you can use the Direct Editor to see the internal name of the property in the **Property** table of the DIM project.

Identifying DIM Elements in an Installation Project



Project • *This information applies to Basic MSI projects.*

When you create a DIM project in InstallShield, InstallShield assigns the project a GUID. This GUID helps to differentiate separate DIM projects. InstallShield also includes this GUID throughout various areas of the DIM project and any Basic MSI project that contains the DIM to identify the data as originating in the DIM.

For example, if you include a file called **MyFile.exe** in a DIM project that has a GUID of **8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC**, InstallShield writes the following value for this file in the **File** table of the DIM project:

MyFile.exe.8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC

If you include that same **MyFile.exe** source file in a different DIM project, InstallShield would append a different GUID to the end of the file's **File** table entry. If you import the DIM project into a Basic MSI project, InstallShield also imports this file-GUID entry into the **File** table of a Basic MSI project. If you add **MyFile.exe** directly to a Basic MSI project, InstallShield does not append a GUID to the end of the file name. If you import the DIM that contains **MyFile.exe** into a Basic MSI project, you will see the

InstallShield uses the DIM GUID for files, components, path variables, and other types of installation data. This helps to make it easy to identify which parts of a Basic MSI project are included directly in the Basic MSI project, and which originate from various DIM projects that are referenced in or imported into the Basic MSI project.

The use of GUIDs also helps prevent conflicts. For example, if you use different values for a specific path variable in multiple DIM projects, import those DIMs into a Basic MSI project that uses another different value for the path variable, you will not encounter any conflicts. InstallShield uses the appropriate values for each path variable instance when building your Basic MSI installation.

Overriding Path Variables in a DIM Project for Use in an Installation Project



Project • *This information applies to Basic MSI projects.*

Depending on how the author of a DIM project defined the project's path variables, it may be necessary for you to override their values when you build a Basic MSI release that includes that DIM. Otherwise, when you build the installation, you may encounter errors because of missing files, or the wrong files may be included in your build.

InstallShield offers several ways for overriding the path variables in a DIM project:

- If you are building a Basic MSI release and you want to override a path variable for a DIM that was imported into the Basic MSI project, you can use the Path Variable Overrides setting on the Build tab in the Releases view.
- If you are building from the command line with ISCmdBld.exe, use the `-l` parameter to specify a path variable name and new value. Use this method if you are building an installation for a Basic MSI project that contains one or more DIMs.
- If you are building through MSBuild or Team Foundation Server (TFS), use the PathVariables parameter on the InstallShield task. This parameter is exposed as the ItemGroup InstallShieldPathVariableOverrides when the default targets file is used.
- You can open the DIM project and update values of path variables as needed. To learn how to open the DIM project from within an open Basic MSI project, see Opening a Referenced DIM Project from Within an Installation Project.

At build time, InstallShield (or the Standalone Build) uses the path variable overrides that you set when obtaining the source files for your project and building the release.

For more information as well as tips on using path variables in DIM projects, see Using Path Variables in a DIM.

Configuring the Target System



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

In addition, certain portions do not apply to certain types of project, as noted.

Every installation changes the target system in some way. The simplest installations might only copy files. More in-depth installations make registry changes, edit .ini files, create shortcuts, configure ODBC resources, use environment variables, modify XML files, modify text files, schedule tasks, and install and control Windows services. For more information on how to configure the target system, refer to this section of the documentation.

Creating Shortcuts and Program Folders



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You can create shortcuts and program folders on the Windows desktop and on the Start menu by using the Shortcuts explorer in the Shortcuts view. Like files, shortcuts are associated with components. In installation projects, they are created on the target system only if the feature to which the component belongs is selected for installation.

Types of Shortcuts



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield offers several types of shortcuts:

Table 4-1 • Types of Shortcuts

Shortcut Type	Project Type	Description
New Shortcut	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Creates a new shortcut to a file that exists in your project. ***Note*** *• This option is disabled when there is no file specified for a component.*
New Advertised Shortcut	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Creates an advertised shortcut. The component's files are not installed to the target system until the end user launches the shortcut.
New Internet Shortcut	InstallScript	Creates an Internet shortcut (one whose Internet Shortcut setting is set to Yes) with a default value of **http://www.YourCompanyName.com** for the Target setting; change this value to the desired URL.
New Shortcut to Preexisting File	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Creates a shortcut to a file that already exists on the target system.

Table 4-1 • Types of Shortcuts (cont.)

Shortcut Type	Project Type	Description
New Folder	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Creates a program folder. You can create a program folder if you want your shortcut to appear under your company name, for example.

Creating Shortcuts



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Before you create a shortcut, you should create at least one feature and component. If you have not created a component when you create your first shortcut, InstallShield creates a component for you. You can change the component to which the shortcut belongs by configuring the shortcut's Component setting.



Task ***To create a shortcut:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, right-click one of the destination directories and then click the appropriate command. For a list of available commands, see Types of Shortcuts.

   InstallShield adds a new shortcut with the default name **NewShortcutN** (where *N* is a successive number).
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.
4. Configure the shortcut's settings.

For details about each of the settings that you can configure for shortcuts and folders, see:

- Shortcut Settings
- Folder Settings

Note *• You can create a program folder if you want your shortcut to appear under your company name, for example. When you have created a folder for your shortcut, you can create the shortcut by right-clicking your new folder and then clicking New Shortcut.*

You cannot create shortcuts to a dynamically linked file. For more information, see Limitations of Dynamic File Linking.

Tip *• You can also use the Components view or—in installation projects only—the Setup Design view to create a shortcut. If you use either the Components view or the Setup Design view, click the Shortcuts item in the explorer; a separate Shortcuts explorer opens in a new pane.*

Configuring Shortcut Settings

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you have created a shortcut, you need to configure its settings. Shortcut settings enable you to link your shortcut to a file, provide a description of the shortcut, or pass arguments.

Task

To configure the settings for a shortcut:

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, click the shortcut. The shortcut's settings are displayed in the right pane.
3. Configure the settings as needed.

For details about each of the settings that you can configure for shortcuts and folders, see:

- Shortcut Settings
- Folder Settings

Specifying the Icon for a Shortcut

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield enables you to specify the icon that should be used for a shortcut that is created on the target system at run time.

Task ***To specify the icon for a shortcut:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, click the shortcut whose icon you want to specify. The shortcut's settings are displayed in the right pane.
3. In the **Icon File** setting, specify the file that contains the icon for the shortcut that you are creating. You must specify an .ico file or the executable file (.dll or .exe) that contains the icon resource.

   For Basic MSI, InstallScript MSI, or Merge Module projects: You can either type the fully qualified path to the file that contains the icon, or click the ellipsis (...) button to browse to it.

   For InstallScript projects: Type the fully qualified path to the file on the target system that contains the icon.
4. If the icon file that you specify contains more than one icon resource, enter the index in the **Icon Index** setting.

   A nonnegative integer refers to the order of the icon resources in the executable file. For example, 0 refers to the first icon in the file, 1 refers to the second icon, and 2 refers to the third icon.

InstallShield changes the icon that is displayed for the shortcut in the Shortcuts explorer to the one that you specified, unless either of the following conditions is true:

- The project type is InstallScript.
- The shortcut is for a pre-existing file on the target system.

For both of the aforementioned conditions, the icon file is not known until run time. Therefore, InstallShield shows the following icon for each shortcut in the Shortcuts explorer, instead of displaying the icon that is used at run time on target systems.



InstallShield also uses this icon for a shortcut in the Shortcuts explorer if the file that is selected in the Icon File setting does not contain an icon.



Project *• For Basic MSI, InstallScript MSI, and Merge Module projects: Since Windows Installer requires a separate icon when the component is advertised, InstallShield extracts the icon from any executable file that you specify.*



Tip *• For Basic MSI, InstallScript MSI, and merge module projects: You can also right-click the icon in the Shortcuts view and then click Change Shortcut icon to specify a different shortcut icon. InstallShield updates the value in the Icon settings with the values that you specify using this method.*

Creating Internet Shortcuts



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

An Internet shortcut is a text file with the .url extension.



Task ***To create an Internet shortcut that opens your Web site:***

1. Create a text file called **MySite.url** with the following contents:

   [InternetShortcut]

   URL=http://www.MySite.com

2. Add this file to a component.

Otherwise, if you simply right-click and click Add, you will add what the shortcut points to and not the shortcut itself.

When the end user launches the shortcut, the specified Web site (**www.MySite.com**) opens in the default browser.

Creating Shortcuts to Folders



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield lets you create a shortcut to a folder. The method that you use to create a shortcut to a folder depends on the project type that you are using.

Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform Projects

The Target setting of a shortcut can contain a directory identifier inside square brackets. For example, to create a shortcut to INSTALLDIR, create a shortcut with the following settings:

- Display Name: Shortcut to INSTALLDIR
- Advertised: No
- Target: [INSTALLDIR]

InstallScript and InstallScript Object Projects

The **AddFolderIcon** function creates a shortcut to a folder if you pass the path to the folder in the third (`szCommandLine`) parameter.

For example, this code creates a shortcut to the Common Files folder in the end user's Program Files folder:

```
ProgDefGroupType(COMMON)

szFolder = TARGETDIR;
LongPathToQuote(szFolder, TRUE);

AddFolderIcon(
    ProgramMenuFolder,        // where shortcut will appear
    "Shortcut to TARGETDIR",  // shortcut display name
    szFolder,        // what shortcut launches
    ""// working directory
    TARGETDIR ^ "Sample.exe", 0,       // icon file, index
    "",    // shortcut key
    NULL);              // special settings
```

InstallScript MSI Project

Use the same basic procedure as in the preceding InstallScript section, changing all occurrences of TARGETDIR to INSTALLDIR in the example code.

Specifying a Keyboard Shortcut for Accessing a Shortcut



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Keyboard shortcuts—also sometimes called hot keys—enable you to perform tasks quickly by pressing a combination of keys, such as CTRL+ALT+A, instead of using the mouse. You can assign a keyboard shortcut to your product's shortcut so that end users can press the appropriate hot keys to launch the shortcut.



***Caution** • It is recommended that you avoid configuring keyboard shortcuts for your shortcuts because they may conflict with existing keyboard shortcuts on the target system.*



Task ***To assign a keyboard shortcut to a shortcut in your project:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, select the shortcut for which you are specifying a hot key.
3. In the **Hot Key** setting, click the ellipsis button (**...**). The **Hot Key** dialog box opens.
4. Press the keyboard shortcut that you want to use for this shortcut.
5. Click **OK**.

In the Hot Key setting, InstallShield displays the appropriate decimal value that represents the combination of keys that you pressed.

For example, if your key combination is CTRL+ALT+A, InstallShield displays 1601 in this setting. This number is obtained by combining the hex value of CTRL (200) and the hex value of ALT (400) with the logical Or operator. Then, that number (600) is added to the hex value for the A key (41) and converted to a decimal value. In this example, the number that is converted to decimal is 641. After the conversion, it is 1601.

Renaming Shortcuts



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

- *MSI Database*
- *MSM Database*
- *Transform*

When you create a shortcut, your shortcut appears with a default internal name. This name is not displayed to end users, but you can change it to something that is relevant to your project.



Task ***To rename a shortcut:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, right-click the shortcut that you want to rename and click **Rename**.
3. Type the new name.

Setting Shell Properties for a Shortcut



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield lets you specify one or more shortcut properties that you want the Windows Shell to set at installation run time. For example, InstallShield has built-in support for enabling you to set Shell properties that control the following behavior:

- Suppressing Initial Pinning of a Shortcut to the Windows 8 Start Screen
- Specifying Whether End Users Should Be Able to Pin a Shortcut to the Taskbar or Start Menu
- Preventing a Shortcut on the Start Menu from Being Highlighted as Newly Installed

InstallShield also lets you set additional properties that the Windows Shell supports. To learn more, see Setting Custom Shell Properties.

Suppressing Initial Pinning of a Shortcut to the Windows 8 Start Screen



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Project-specific differences are noted where applicable.

InstallShield lets you specify whether you want to suppress the initial pinning of a shortcut to the Start screen on Windows 8 target systems.



***Note** • Windows 8.1 disabled automatic pinning of freshly installed applications/shortcuts.*

If you specify that you want shortcut pinning to be suppressed, the installation sets a Windows Shell property that was introduced in Windows 8. You may want to disable pinning for shortcuts that are for tools and secondary products that are part of your installation.

Earlier versions of Windows ignore this shortcut property.



Task ***To specify whether pinning a shortcut to the Windows 8 Start screen should be suppressed by default:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, select the shortcut that you want to configure. The shortcut's settings are displayed in the right pane.
3. In the **Shell Properties** setting, click the **Add New Shell Shortcut Property** button, and then click **Suppress Initial Pin to Windows 8 Start Screen**. InstallShield adds a new **Key Name** setting, plus additional rows of related settings under it, and configures them as needed.

Windows Installer 5 introduced support for shortcut properties. Earlier versions of Windows Installer ignore these setting.

Note that Windows 8 maintains information about shortcut pinning to the Start screen after a shortcut is removed by uninstalling the application. Therefore, this setting has no effect on the target system if the shortcut has already been installed on it. Thus, when you are testing this functionality, ensure that you test on a clean machine—one on which this shortcut and its target have never been installed.

For Windows Installer–based projects (Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform), some end users have reported run-time warnings or errors with a message such as the following one when Windows Installer is installing a shortcut that configures Shell properties:

```
Property [1] for shortcut '[2].lnk' could not be set.
```

In the aforementioned message, [1] is the name of the Shell property that Windows Installer is attempting to set, and [2].lnk is the name of the shortcut file. In such cases, it appears that the .lnk file could be locked by a different process.

InstallScript Language Support

The following InstallScript functions enable you to prevent the pinning of a shortcut to the Start screen:

- CreateShortcut
- SetShortcutProperty

Specifying Whether End Users Should Be Able to Pin a Shortcut to the Taskbar or Start Menu



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Project-specific differences are noted where applicable.

InstallShield lets you specify whether you want the context menu commands for pinning a shortcut to the taskbar and to the Start menu are to be displayed after end users install your product. You may want to disable pinning for shortcuts that are for tools and secondary products that are part of your installation. Note that if you configure the shortcut to prevent this pinning, the target of the shortcut is ineligible for inclusion in the most frequently used list on the Start menu.

If you configure a shortcut to prevent this pinning functionality, the installation sets a Windows Shell property. By default, the property is not set for new shortcuts, so end users are able to pin the shortcut to the taskbar and to the Start menu.

Note that shortcuts that contain certain strings cannot be pinned to the taskbar or the Start menu, and they cannot be displayed in the most frequently used list. Examples are:

- Documentation
- Help
- Install
- Remove
- Setup
- Support

Note that the method for hiding the context menu commands varies, depending on the project type that you are using.

Instructions for Windows Installer–Based Projects

For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects, use the Shell Properties setting to hide the context menu commands.



Task ***To hide the context menu commands for pinning a shortcut to the taskbar or Start menu:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, select the shortcut that you want to configure. The shortcut's settings are displayed in the right pane.
3. In the **Shell Properties** setting, click the **Add New Shell Shortcut Property** button, and then click **Prevent Pinning**. InstallShield adds a new **Key Name** setting, plus additional rows of related settings under it, and configures them as needed.

To allow the context menu commands to be displayed, find the Key Name setting that has a Property subsetting with either of the following, and click the **Delete this shortcut shell property** button in the Key Name setting:

- System.AppUserModel.PreventPinning
- {9F4C2855-9F79-4B39-A8D0-E1D42DE1D5F3}, 9

Windows Installer 5 introduced support for this shortcut property. Earlier versions of Windows Installer ignore this setting.

Note that for Windows Installer–based installations, some end users have reported run-time warnings or errors with a message such as the following one when Windows Installer is installing a shortcut that configures Shell properties:

```
Property [1] for shortcut '[2].lnk' could not be set.
```

In the aforementioned message, [1] is the name of the Shell property that Windows Installer is attempting to set, and [2].lnk is the name of the shortcut file. In such cases, it appears that the .lnk file could be locked by a different process.

Instructions for InstallScript Projects

For InstallScript projects, use the Prevent Pinning setting.



Task ***To hide the context menu commands for pinning a shortcut to the taskbar or Start menu:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, select the shortcut that you want to configure. The shortcut's settings are displayed in the right pane.
3. In the **Prevent Pinning** setting, select **Yes**.

Windows 7 introduced support for this setting. Earlier versions of Windows ignore this setting.

InstallScript Language Support

The following InstallScript functions enable you to hide the context menu commands for pinning a shortcut to the taskbar or the Start menu:

- CreateShortcut
- SetShortcutProperty

Preventing a Shortcut on the Start Menu from Being Highlighted as Newly Installed



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Project-specific differences are noted where applicable.

InstallShield lets you optionally prevent a shortcut on the Start menu from being highlighted as newly installed after end users install your product. This has the same effect as clearing the **Highlight newly installed programs** check box in the Customize Start Menu dialog box for an individual item on a target system. You may want to set this property for shortcuts that are for tools and secondary products that are part of your installation.

If you specify that you want to prevent the highlight functionality for a shortcut, the installation sets a Windows Shell property. By default, the property is not set for new shortcuts.

Note that the method for preventing the highlight behavior varies, depending on the project type that you are using.

Instructions for Windows Installer–Based Projects

For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects, use the Shell Properties setting to prevent the highlight behavior.



Task ***To prevent the Start menu entry for a shortcut from being highlighted as newly installed:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, select the shortcut that you want to configure. The shortcut's settings are displayed in the right pane.
3. In the **Shell Properties** setting, click the **Add New Shell Shortcut Property** button, and then click **Do Not Highlight as New**. InstallShield adds a new **Key Name** setting, plus additional rows of related settings under it, and configures them as needed.

To enable the Start menu entry to be highlighted, find the Key Name setting that has a Property subsetting with either of the following, and click the **Delete this shortcut shell property** button in the Key Name setting:

- System.AppUserModel.ExcludeFromShowInNewInstall
- {9F4C2855-9F79-4B39-A8D0-E1D42DE1D5F3}, 8

Windows Installer 5 introduced support for this shortcut property. Earlier versions of Windows Installer ignore this property.

Note that for Windows Installer–based installations, some end users have reported run-time warnings or errors with a message such as the following one when Windows Installer is installing a shortcut that configures Shell properties:

```
Property [1] for shortcut '[2].lnk' could not be set.
```

In the aforementioned message, [1] is the name of the Shell property that Windows Installer is attempting to set, and [2].lnk is the name of the shortcut file. In such cases, it appears that the .lnk file could be locked by a different process.

Instructions for InstallScript Projects

For InstallScript projects, use the Do Not Highlight as New setting.



Task ***To prevent the Start menu entry for a shortcut from being highlighted as newly installed:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, select the shortcut that you want to configure. The shortcut's settings are displayed in the right pane.
3. In the **Do Not Highlight as New** setting, select **Yes**.

Windows 7 introduced support for this setting. Earlier versions of Windows ignore this setting.

InstallScript Language Support

The following InstallScript functions enable you to prevent the highlight behavior:

- CreateShortcut
- SetShortcutProperty

Setting Custom Shell Properties



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Project-specific differences are noted where applicable.

InstallShield lets you specify one or more shortcut properties that need to be set by the Windows Shell at run time. The properties that the Shell can set are defined in **propkey.h**, which is part of the Windows SDK.

Note that the method for preventing the highlight behavior varies, depending on the project type that you are using.

Instructions for Windows Installer–Based Projects

For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects, use the Shell Properties setting to set one or more Shell properties.



Task ***To set a Shell property for a shortcut that is in your project:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, select the shortcut that you want to configure. The shortcut's settings are displayed in the right pane.
3. In the **Shell Properties** setting, click the **Add New Shell Shortcut Property** button, and then click **Custom Property**. InstallShield adds a new **Key Name** setting, plus additional rows of related settings under it, with placeholder text.
4. Configure each of the settings as needed.

You can add multiple properties for the shortcut if necessary.

Windows Installer 5 introduced support for the Shell shortcut properties. Earlier versions of Windows Installer ignore these properties.

For more information about configuring shortcut properties, see MsiShortcutProperty Table in the Windows Installer Help Library.

Note that for Windows Installer–based installations, some end users have reported run-time warnings or errors with a message such as the following one when Windows Installer is installing a shortcut that configures Shell properties:

```
Property [1] for shortcut '[2].lnk' could not be set.
```

In the aforementioned message, [1] is the name of the Shell property that Windows Installer is attempting to set, and [2].lnk is the name of the shortcut file. In such cases, it appears that the .lnk file could be locked by a different process.

InstallScript Language Support

The following InstallScript functions enable you to configure Shell properties in InstallScript projects and in other project types:

- CreateShortcut
- SetShortcutProperty

Creating Uninstallation Shortcuts for Basic MSI Projects



Project *• This information applies to Basic MSI projects.*



Windows Logo *• According to current Windows logo guidelines, best practice is to not place shortcuts to remove the application in the Start menu. An uninstallation shortcut is unnecessary because your application's uninstaller is in Add or Remove Programs.*

You can create an uninstallation shortcut to make it easier for end users to uninstall your product from their systems. When launched, the shortcut automatically starts the uninstallation process.

Task ***To create an uninstallation shortcut:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, right-click the destination directory that should contain the uninstallation shortcut, and then click **New Shortcut to Preexisting file**. InstallShield adds a new shortcut with the default name **NewShortcutN** (where *N* is a successive number).
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.
4. In the **Arguments** field, type **/x [ProductCode]**. Separate the two arguments with a space.

   Msiexec.exe is the command-line engine for the Windows Installer service. The `/x` argument instructs the Windows Installer service to uninstall the product referenced by the product code.
5. Set the **Advertised** field to **No**.
6. In the **Target** field, select **[SystemFolder]**, and then append **msiexec.exe** to this location.
7. In the **Component** field, select the component with which you want this shortcut associated. If the shortcut should always be installed, associate it with the component containing your application's main executable file.

Creating Uninstallation Shortcuts for InstallScript and InstallScript MSI Projects

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Note *• You may not want to create an uninstallation shortcut. An uninstallation shortcut is unnecessary because your application's uninstaller is in Add or Remove Programs.*

You can create an uninstallation shortcut to make it easier for end users to uninstall your product from their systems. When end users launch the shortcut, the uninstallation process automatically starts.

The following InstallScript code creates a shortcut called **Uninstall Application** on the desktop after file transfer. Launching this shortcut runs your installation in maintenance mode. This code works for both InstallScript and InstallScript MSI projects.

```
function OnMoved()
begin
   if( !REMOVEALLMODE ) then
      AddFolderIcon( FOLDER_DESKTOP, "Uninstall Application", UNINSTALL_STRING +
ADDREMOVE_STRING_REMOVEONLY, "", "", 0, "", REPLACE );
   endif;
end;
```

Tip • *If you include ADDREMOVE_STRING_REMOVEONLY,* `-removeonly` *is added to the command line. As a result, REMOVEONLY is set, and the OnMaintUIBefore event handler shows only the uninstallation option. To display the standard maintenance user interface, do not include ADDREMOVE_STRING_REMOVEONLY.*

Configuring the Appearance of a Desktop App's Tile on the Start Screen

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Windows 8 introduced a grid of application tiles to the Start screen, replacing the usual list of shortcuts, and also presented tiles in place of shortcuts. InstallShield supports customizing the appearance of a desktop app's tile on the Start screen. The following tile configuration settings are available:

- A toggle between light or dark text when including the app name on medium-sized (150x150) tiles
- Choice of tile background color
- Option to use custom tile images (small: 70x70 and medium:150x150)
- Preference to show or hide the app name on medium-sized tiles

The **Tile Configurations** node appears in the main **Shortcuts** view and in each component's **Shortcuts** subview. Any applicable tile configurations are listed. For a component, **Tile Configurations** only applies if it is a configuration for an .exe file in that component. For the main view, it only applies if the filtering includes the component targeting the configured .exe file.

Note • *While there is a relation between shortcuts and tiles, they are different. You can configure shortcuts individually, and even create multiple shortcuts to the same .exe file that use different icons; however, you can only create one tile configuration per .exe file.*

Task ***To configure the appearance of a desktop app's tile on the Start screen:***

1. In the **Shortcuts** view, or in the **Shortcuts** node of the **Setup Design** or **Components** view, click **Shortcuts.**
2. Right-click **Tile Configurations** and choose the **Add Tile Configuration** context menu option.



***Note** • This step assumes an .exe file has been added to the project. If an .exe file does not exist in the project, the **Add Tile Configuration** context menu option is disabled.*



***Tip** • Alternatively, you can right-click the shortcut that targets the .exe file for which you want to configure a Start screen tile and then choose the **Configure Tile** context menu option. A new tile configuration is added (and selected). If a configuration has already been added for the shortcut, the **Configure Tile** option is disabled.*

3. In the resulting Browse for Tile Target dialog box, navigate to and select the .exe file for the app whose tile appearance you want to configure. A new tile configuration is added.
4. Use any of the settings available for **Tile Configurations** to customize the appearance of a desktop app's tile on the Start screen. For additional information regarding available settings, refer to Tile Configuration Settings.

Editing the Registry



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

The Windows registry is a system-wide database that contains configuration information used by applications and the operating system. The registry stores all kinds of information, including the following:

- Application information such as company name, product name, and version number
- Path information that enables your application to run
- Uninstallation information that enables end users to uninstall the application easily without interfering with other applications on the system
- System-wide file associations for documents created by an application
- License information
- Default settings for application options such as window positions

Keys, Value Names, and Values

The registry consists of a set of keys arranged hierarchically under the My Computer explorer. Just under My Computer are several root keys. An installation can add keys and values to any root key of the registry. The root keys that are typically affected by installations are:

- HKEY_LOCAL_MACHINE
- HKEY_USERS
- HKEY_CURRENT_USER
- HKEY_CLASSES_ROOT

A key is a named location in the registry. A key can contain a subkey, a value name and value pair, and a default (unnamed) value. A value name and value pair is a two-part data structure under a key. The value name identifies a value for storage under a key, and the value is the actual data associated with a value name. When a value name is unspecified for a value, that value is the default value for that key. Each key can have only one default (unnamed) value.

Note that the terms key and subkey are relative. In the registry, a key that is below another key can be referred to as a subkey or as a key, depending on how you want to refer to it relative to another key in the registry hierarchy.

InstallShield Projects and the Registry

InstallShield includes the Registry view to help you with the task of modifying the end user's registry. Use this view to create keys and values in much the same way that you use the Windows Registry Editor.

All registry data (except the <Default> registry set in a InstallScript project) must be associated with a component. In installation projects, if the component's feature is selected for installation, the component's registry data are set up on the target system.



Caution *• It is important not to modify or delete registry keys indiscriminately because the registry is a vital part of the Windows operating system, and the system may fail to function if vital registry keys are altered.*



Note *• The installer automatically creates certain registry entries based on values that you provide in the General Information view.*



Windows Logo *• These "informational keys" are required if you want to meet the requirements for the Windows logo program.*

Also, all of a component's advanced settings are used to register files on the target system.

Filtering Registry Entries by Component or Feature



Project *• This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Several project types (DIM, Merge Module, and MSM Database) do not include support for features; these project types have support for by filtering by component, but not by feature.

The Registry view includes a View Filter. This filter enables you to select a component or feature whose registry data you want to display in the view.

The View Filter lists your project's hierarchy of features, subfeatures, and components. Selecting a feature shows the registry data for all of the components in that feature, but it allows you only to modify and delete existing entries. You need to select a component in order to add a new registry key.

If the feature hierarchy contains a subfeature, selecting a parent feature displays only the registry entries in that feature. It does not display the registry entries in the subfeature.

Browsing for Components

You can browse for components from within the registry explorer. Click the ellipsis button to the right of the View Filter to launch the Browse for Component dialog box. From that dialog box, you can select a feature in the Feature list and then select an existing component or click the new component button to create a new component.

If the View Filter is set to All Application Data, you can right-click a registry hive in the Destination computer's registry view pane and click Browse for Component to display the dialog box.

Viewing All Registry Entries in Your Project

To view all of the registry entries in your installation, select the All Application Data option in the View Filter. You can export registry data by selecting Export All or Export Selected Branch from the context menu after right-clicking a registry key.



***Note** • You can modify, rename, or delete registry keys and values while filtering the view by All Application Data.*

When you click a registry key in the Destination computer's Registry view pane of the Registry view, InstallShield displays all of the registry data for that key in the lower-right pane of the Registry view. The Component column shows the component with which the registry data is associated. If the same value exists for more than one component, the last component that was associated with the registry data is displayed.

If you have not set a value for a key, no registry data is displayed for that key when you select All Application Data in the View Filter.

Refreshing the Registry View



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

Task ***To refresh the Registry view:***

Press F12.

Creating Registry Keys

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

Task ***To specify a registry key to be created on the target system when a component is installed:***

1. In the View List under **System Configuration**, click **Registry**.
2. *In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform, and QuickPatch projects*—In the **View Filter** list, click the component with which you want to associate the registry data.

   Note *• If* ***All Application Data*** *is selected, the* ***Registry*** *view is read-only.*

   In InstallScript and InstallScript Object projects—In the **Destination computer's Registry view** pane, open an existing registry set or create one by right-clicking the **Destination Computer** folder. Associate the registry set with one or more components by clicking the registry set and selecting the desired components in the **Registry Set Install Conditions** pane.
3. In the **Destination computer's Registry view** pane, right-click a registry hive or key, point to **New**, and then click **Key**. InstallShield adds a new key with the name **New Key-n** (where *n* is a successive number).

4. Enter a meaningful name to rename the key, or right-click the key and click **Rename** to give it a new name later.

InstallShield adds your new key with an empty default string value.

By default, all keys that you create are set for automatic installation and uninstallation. This means that they are installed if the component they belong to is installed, and they are uninstalled when that component in uninstalled. For more information on registry key flags, see Registry Flags.

Tip • *You can create multiple nested keys at one time by creating a new key and typing, for example,* **Key 1\Key 2\Key 3** *in the key's name. InstallShield creates a nested key structure where Key 3 is a subkey of Key 2, which is a subkey of Key 1.*

Dragging and Dropping Registry Entries to Create Registry Keys

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

The quickest way to add registry entries to your installation project is to drag them from one of the source panes in the Registry view and drop them into one of the destination panes. When you drop an entire key onto the **Destination computer's Registry view** pane, all of that key's subkeys and values are added to the selected component or, in a InstallScript project, to the registry set on which you drop the entry.

Using the Context Menu to Drag and Drop Keys

You can use the context menu to move multiple keys and values at one time. Right-click a registry entry, drag it to a destination, and click a command on the context menu.

Table 4-2 • Commands Available from the Registry Entry Context Menu

Option	Description
All keys & values	Adds all selected keys, subkeys, and values.
Key and its values only	Adds only the selected key and the key's values. No subkeys are added.
Only this key	Adds only the selected key, not any of its subkeys or values.

Table 4-2 • Commands Available from the Registry Entry Context Menu (cont.)

Option	Description
Cancel	Ends the drag and drop operation without making any changes.

Viewing Both the 32- and 64-Bit Areas of the Source Machine's Registry on 64-Bit Development Systems

If you are using InstallShield on a 64-bit development system, the Registry view in InstallShield displays both the 32-bit and 64-bit areas of your machine's registry:

- HKEY_LOCAL_MACHINE\Software
- HKEY_LOCAL_MACHINE\Software\Wow6432Node

This support enables you to drag and drop entries from those source areas to the appropriate areas in the destination pane of this view when you are configuring registry data changes for your project.

Note that if you want your installation to install registry data to a 64-bit area of the registry on 64-bit target systems without having it redirected to a 32-bit area, you must place the registry data in a component that is marked as 64 bit. Simply dragging 64-bit data from the source panes in the Registry view to a location in one of the destination panes of the view does not mark the component as 64 bit. For more information, see Targeting 64-Bit Operating Systems.

Importing Data From Another Machine

A limitation of the drag-and-drop procedure is that it works only if the registry entries exist on your installation development system. If you have registry data from another machine, you can import that data with the Import REG File Wizard.

Removing Registry Keys



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

Task ***To remove a registry key:***

1. In the View List under **System Configuration**, click **Registry**.
2. *In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform, and QuickPatch projects*—In the **View Filter** list, click the component that contains the registry key, or select **All Application Data** to view all of the registry keys for your product.

   In InstallScript and InstallScript Object projects—In the **Destination computer's Registry view** pane, open the existing registry set that contains the registry key.
3. In the **Destination computer's Registry view** pane, right-click the registry key that you want to remove and then click **Delete**.

Creating Registry Values

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

Task ***To specify a registry value to be created on the target system:***

1. In the View List under **System Configuration**, click **Registry**.
2. In the **Destination computer's Registry view** pane, click the key to which you want to add a value. All existing registry values for that key are listed in the **Destination computer's registry data** pane.
3. Right-click in the list of values and then click the type of data you want to register. Available options are as follows:

   Table 4-3 • Types of Registry Values

Option	Description
Default Value	The key's default value.
String Value	A fixed-length text string.

Table 4-3 • Types of Registry Values (cont.)

Option	Description
Binary Value	The value is interpreted and stored as a hexadecimal value.
DWORD Value	Data represented by a number that is four bytes (32 bits) long.
Multi-String Value	Multiple text strings formatted as an array of null-terminated strings, and terminated by two null characters. Selecting this command launches the Multi-Line String Value dialog box.
Expandable String Value	The value is interpreted and stored as an expandable string. According to the Microsoft Developer Network (MSDN), an expandable string registry value is a null-terminated string that contains unexpanded references to environment variables (for example, **%PATH%**).

InstallShield adds a new value with the name New Value n (where *n* is a successive number). Enter a meaningful name now to rename the value, or right-click the value name and then click Rename to give it a new name later.

Project • *In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform, and QuickPatch projects, any registry value can serve as the component's key path.*

Modifying Registry Value Data

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

Task ***To modify the data for a registry value to be created on the target system:***

1. In the View List under **System Configuration**, click **Registry**.
2. In the **Destination computer's Registry view** pane, click the registry key that has the value that you want to modify. All registry values are listed in the **Destination computer's registry data** pane.

3. Double-click the value that you want to modify. The Edit Registry Data dialog box or the Multi-Line String Value dialog box opens.
4. Complete the information in the dialog box, and then click **OK**.



***Project** • In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform, and QuickPatch projects—To add value data that contains square brackets ([]), you must precede each bracket with a backslash (\) and surround it with an opening and closing bracket. Otherwise, Windows Installer treats the value as a property. For example, if you wanted to write **[stuff]** to the registry, you would use **[\[]stuff[\]]** as the value data.*

Removing Registry Values



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*



Task ***To remove a registry value from your project:***

1. In the View List under **System Configuration**, click **Registry**.
2. In the **Destination computer's Registry view** pane, click the registry key that has the value that you want to delete. All registry values are listed in the **Destination computer's registry data** pane.
3. In the **Destination computer's Registry data** pane, right-click the registry value that you want to remove and then click **Delete**.

Registry Flags



***Project** • Support for registry flags differs, depending on what project type you are using.*

Registry flags enable you to control the installation of your registry entries.

Registry Flags in Windows Installer–Based Projects



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

By default, your registry entries are created on the target system if the component to which they belong is installed. They are then removed from the target system when that component is removed. If you would like your registry entries to remain on the target system even after the product has been uninstalled, or if you want to create registry entries only if they do not already exist, you need to set the installation flag for that key.

In InstallShield, installation behavior is set at the subkey level. All values beneath the key must have the same installation and uninstallation behavior.

To change the registry flag of a key, right-click one of your project's keys in the Registry view, and then click any of the commands that are listed in the following table.

Table 4-4 • Types of Registry Flags in Windows Installer–Based Projects

Icon	Command	Description
	Automatic	This is the default option for all registry keys. If the key is not already present, the installation creates it. During an uninstallation, if the key is empty, it is removed.
	Install only (+)	If the key does not already exist, it is created. The key is left on the target system when the component to which this key belongs is uninstalled. This option is available only for keys that do not contain subkeys or values.
	Uninstall entire key (–)	If this flag applies to an empty key, the key is not created at installation. If this flag applies to a key that contains values and the key does not exist on the target system, the key and values are created at installation. In both cases, the key, all subkeys, and values are removed at uninstallation—even if the subkeys and values were added after the installation.
	Install if absent, Uninstall if present (*)	This option is similar to the default behavior (the Automatic registry flag), with one exception. For the Automatic registry flag, if the key is not empty during uninstallation, it is not removed. For the **Install if absent, Uninstall if present (*)** flag, if the registry key is present during uninstallation, the key is removed, regardless of whether its subkeys or values remain.

Registry Flags in InstallScript-Based Projects

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

After your installation has created registry keys on the target system by installing a registry set, subkeys or values may be created under those keys by other installations or applications (or directly by the end user). You can prevent a registry key that your installation installed from being uninstalled if it has any subkeys or values that were not created by the installation. To accomplish this, you can change the registry flag of the key. To do so, right-click one of your project's keys in the Registry view and then click the following command.

Table 4-5 • Registry Flag in InstallScript-Based Projects

Icon	Command	Description
	Shared among several applications	When the component to which this key belongs is uninstalled, the key is not removed from the target system if the key has any subkeys or values. (Note that a value that was created by your installation will have already been uninstalled unless you unset its **Remove during Uninstall** flag.) If the key does not have any subkeys or values, the key is removed from the target system regardless of whether it was created by the installation—that is, whether it already existed at the time of installation.

You may want your uninstallation to leave some of your product's registry values on the end user's system—for example, registry values that are used by the operating system and are modified by your installation. To do this, right-click the key that contains the registry value that you do not want to be uninstalled. Select the **Shared among several applications** flag for that key. Then right-click the value of the key and ensure that the **Remove during Uninstall** command is not selected.

Searching for Registry Entries in a Project

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

Task ***To search for a registry entry within a certain component (in Windows Installer projects) or registry set (in InstallScript projects):***

1. In the View List under **System Configuration**, click **Registry**.
2. *In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform, and QuickPatch projects*—In the **View Filter** list, select the component that you would like to search. Note that if **View All Entries** is selected, the **Registry** view is read only.

   In InstallScript and InstallScript Object projects—Open the registry set that you would like to search.
3. Right-click a registry entry and click **Find**. The **Find** dialog box opens.
4. Enter the string that you are searching for.
5. Click **Find** to start the search.

In a Windows Installer project, InstallShield searches only the registry entries for the current component, not all the registry data for the project.

The search starts from the top item and works its way down, regardless of which item is selected. It stops at the first match, highlighting this key. To continue the search, right-click and select Find Next, or press F3. The Find Next feature finds the next match, moving down the explorer.

Setting Uninstall Behavior for Registry Keys

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

In InstallShield, uninstall behavior for registry entries is set at the key level.

Task ***To set the uninstall behavior for a registry key:***

1. In the View List under **System Configuration**, click **Registry**.
2. In the **Destination computer's Registry view** pane, right-click a key and click the appropriate uninstall behavior.

Uninstall Behavior Options

In InstallShield, installation behavior is set at the subkey level. All values beneath the key must have the same installation and uninstallation behavior. For a list of available options, see Registry Flags.

Using Environment Variables in Registry Values



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

With REG_EXPAND_SZ string values, you can use environment variables for paths that are stored in the registry. These entries require special formatting in order to be recognized by the operating system as environment variables. The format for a REG_EXPAND_SZ value as it appears in the registry is %TEMP%. TEMP is the standard environment variable for the TEMP directory.

Syntax

When creating registry entries of this type, you need to begin the entry with a pound sign followed by a percent sign (#%). Then, you can enter your environment variable, complete with the beginning and ending percent sign. Therefore, if you enter **#%%TEMP%** in the Registry view in InstallShield, it appears as **%TEMP%** when written to the registry.

The Type field for this entry appears as REG_EXPAND_SZ, and the Data field is %TEMP%.

Writing Property Values to the Registry



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

For Windows Installer–based projects, you can use Windows Installer properties in registry values to store information for later use by your product. At run time, Windows Installer automatically expands expressions of the form **[PropertyName]** in registry data to the value of the property called **PropertyName**. Any property defined in the Property Manager view can be used in this way.

For example, if you would like to store the destination location of your software, create a registry value whose data is **[INSTALLDIR]**. At run time, when your installation creates the registry value, the data for that registry value is set to the location where your application is installed.

You can use the same format for registry key names and value names. For example, if you want a key name to be the name of your company, enter **[Manufacturer]** for the name of the key. When the registry key is created on the target system, the name of this key will be the value of the Publisher setting, as entered in the General Information view.

Importing Registry Files



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield enables you to import any existing registry (.reg) files that you may have from other installation projects or that you have created outside InstallShield



Task ***To import a .reg file:***

1. Launch the Import REG File Wizard.
2. Follow the instructions in the wizard panels to import your .reg file.

When you import a .reg file into a component or registry set, that registry data is added to the component's or registry set's registry data and written to the end user's system if the component or registry set is installed.

Exporting Registry Files



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you export a registry (.reg) file, you are copying a portion of your program's registry data to a registry file.



Task ***To export a registry file:***

1. In the View List under **System Configuration**, click **Registry**.
2. In the **Destination computer's Registry view** pane, right-click the registry entry that you want to export and click either **Export REG File** or **Export Selected Branch**.
   - **Export REG File** exports the entire component's registry data.
   - **Export Selected Branch** exports only the current registry key and any subkeys.

Setting Key Paths for Components



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

A key path is a unique registry value for each component that the Windows Installer uses to detect the component's presence. This value must contain a file or a folder.



Task ***To set a component's key path:***

1. In the View List under **System Configuration**, click **Registry**.
2. In the **Destination computer's Registry view** pane, click the registry entry that contains the registry data that you want to use for the key path.
3. In the **Destination computer's registry data** pane, right-click the registry key's value name and then click **Set Key Path**.

The value's string, binary, or DWORD icon is replaced with a key icon.

Task ***To remove a component's key path:***

Right-click the key path and click **Clear Key path**.

Note • *A component can have either one key path or one key file. If you have already set a key file or a key path for a component, you will get a warning prompt if you try to set another key path. Click Yes in the dialog box to replace the existing key file or key path with the new key path.*

Configuring Permissions for Registry Keys

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield lets you configure settings for securing registry keys for end users who run your product in a locked-down environment. You can assign permissions for a registry key to specific groups and users. For example, you may assign Read, Write, and Delete permissions for a particular registry key to the Administrators group, but only Read permissions for all of the users in a different group.

Task ***To configure the permissions for a registry key:***

1. In the View List under **System Configuration**, click **Registry**.
2. In the **Destination computer's Registry view** pane, right-click the registry key and then click **Permissions** button. The **Permissions** dialog box opens.
3. Add, modify, and remove permissions entries as needed. For more information, see Permissions Dialog Boxes for Registry Keys.

Depending on what is selected for the Locked-Down Permissions setting in the General Information view of your project, InstallShield adds permissions data to either the **ISLockPermissions** table or the **LockPermissions** table. To learn more, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.

Primary Keys for the Registry Table

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Windows Installer requires a unique primary key for each registry key and value that you add to the **Registry** table. To enable you to create registry entries in a completely visual environment, InstallShield assigns a unique name to every entry in the database's **Registry** table at build time.

You may need to know the entry's primary key when authoring a custom action. InstallShield supports specifying a primary key on a registry key or value in the Registry Data explorer in the Setup Design view or the Components view. Note that it is not possible to assign the key or value a primary key in the Registry view.

Specifying a Primary Key for the Registry Table

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Task ***To specify a primary key for a registry key or value:***

1. Right-click the key or value and click **MSI Value**. The MSI Value dialog box opens.
2. Enter the name of the key. Since the primary key must be a Windows Installer identifier, the name may contain only letters, numbers, underscores (_) and periods (.), and it must begin with a letter or underscore.

To view or modify the primary key, right-click the registry key or value and click MSI Value again.

If you do not specify a value, InstallShield generates a unique primary key for this entry in the **Registry** table.

Entering Multi-Line String Values for Registry Keys

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

The Registry view provides multi-line string support when you want to add a value to a particular registry key.

Task ***To add a multi-line string value:***

1. Right-click the registry key to which you want to add the multi-line value, point to **New**, and click **Multi-String Value**. The Multi-Line String Value dialog box opens.
2. Select how you want to modify the registry value, and then enter a line for each null-delimited string.
3. Click **OK**.

Note *• Strings can contain just spaces, but they cannot be empty or* **[~]***, which is the delimiter for the strings.*

Writing Entries Under HKCU for a Per-User Installation

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

Since the current user may not have sufficient privileges for modifying keys under HKEY_LOCAL_MACHINE, you may need to write the entries under HKEY_CURRENT_USER.

When you select HKEY_USER_SELECTABLE in the Registry view, the entries are created under the appropriate registry hive, according to the type of installation and the end user's access rights. In a per-user installation, which means that the **ALLUSERS** property is set to 2 or that the installation is being run by someone with user-level access privileges, these entries would be made under HKEY_CURRENT_USER. In a per-machine installation, which means that **ALLUSERS** is not null and that the end user is an administrator, the entries would be written under HKEY_LOCAL_MACHINE.

Note • *Windows does not allow a new key to be created directly under HKEY_LOCAL_MACHINE. For this reason, any information that you create under HKEY_USER_SELECTABLE must be placed under the SOFTWARE subkey, which is the only subkey common to HKEY_LOCAL_MACHINE and HKEY_CURRENT_USER.*

For example, creating the key HKEY_USER_SELECTABLE\SOFTWARE\MyCompany is valid, but creating HKEY_USER_SELECTABLE\MyCompany is not.

Setting or Getting Multi-Line Strings in the Registry

Project • *This information applies to InstallScript projects.*

Setting Multi-Line Strings

To set multi-line strings, call the **RegDBSetKeyValueEx** function. The `szValue` parameter should contain the substrings to be added, separated by null characters. You must add two null characters to the end of the main string to denote its end. To create such a string, follow these steps:

1. Initialize the string to contain each of the substrings, separated by space characters.
2. Assign the ASCII value for a null string to the string positions after each line.
3. Assign an additional null character (ASCII value 0) to the end of the string.

For example:

```
svValue = "This is line one. This is line two. This is line three. ";
svValue[17] = 0;
svValue[35] = 0;
svValue[55] = 0;
svValue[56] = 0;
```

Retrieving Multi-Line Strings

To get multi-line strings, call the **RegDBGetKeyValueEx** function. The substrings will be returned in a main string, separated by null characters. To extract them, use the following code to read the strings into a string list and display them in an **SdShowInfoList** dialog:

```
// svValue is the string read in by RegDBGetKeyValueEx
listID = ListCreate( STRINGLIST );
if (listID != LIST_NULL) then
    StrGetTokens( listID, svValue, "" );
    SdShowInfoList( szTitle, szMsg, listID );
endif;
```

Working with Registry Functions



Project • *For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.*

The InstallScript language includes many built-in functions that help you configure an installation so that it accesses the registry; reads, creates, and deletes registry keys; and establishes registry-related parameters for uninstallation.

For example, you can sometimes determine if an application is present on the target system by detecting registry keys used by the application. Many applications store data in the registry key HKLM\SOFTWARE\CompanyName\ProductName\ProductVersion. To detect if a registry key exists, you can call the **RegDBKeyExist** function in your InstallScript code as follows:

```
// always set root key before calling other RegDB functions
RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

if (RegDBKeyExist("Software\\ThisCo\\ThisApp") = 1) then
   MessageBox("ThisApp is on the system.", INFORMATION);
endif;
```

To learn more about the built-in registry functions available with InstallScript, see the following:

- Registry Functions
- Special Registry-Related Functions

Reading Data from the Registry

Basic MSI Project

The **AppSearch** and **RegLocator** tables can read a value from the registry. To read the RegisteredOwner value mentioned above, add the following record to the **AppSearch** table, using the Direct Editor:

Table 4-6 • Values for the AppSearch Table

Field	Value	Comments
Property	REGISTERED_OWNER	Must be a public property
Signature_	registry_sig	

Next, add the following record to the **RegLocator** table:

Table 4-7 • Values for the RegLocator Table

Field	Value	Comments
Signature_	registry_sig	Same signature as above
Root	2	HKEY_LOCAL_MACHINE

Table 4-7 • Values for the RegLocator Table (cont.)

Field	Value	Comments
Key	Software\Microsoft\Windows NT\CurrentVersion	
Name	RegisteredOwner	
Type	2	Registry data

After the AppSearch action runs, the **REGISTERED_OWNER** property will contain the data read from the registry. If the value is not found, the property will be undefined (empty).

For more information regarding the **AppSearch** and **RegLocator** tables, see the Windows Installer Help Library.

For Windows Installer-based projects, you can use the System Search Wizard to find data.

InstallScript MSI Project

With InstallScript, you can use the **RegDBGetKeyValueEx** function. For example, to read the RegisteredOwner value from the key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion, you can use code similar to the following:

```
function readRegisteredOwner( )
    STRING svRegisteredOwner;
    NUMBER nvType, nvSize;
begin
RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

RegDBGetKeyValueEx(
    "Software\\Microsoft\\Windows NT\\CurrentVersion",
    "RegisteredOwner",
    nvType,
    svRegisteredOwner,
    nvSize);

MessageBox(
    "Registered owner is: " + svRegisteredOwner,
    INFORMATION);
end;
```

You can set a Windows Installer property equal to the value you read using the **MsiSetProperty** function.

Changing .ini File Data

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

- *MSI Database*
- *MSM Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes .ini file functions for modifying .ini file data. You can use these functions in InstallScript projects.

Initialization (.ini) files serve as a database in which you can store and retrieve information between the uses of your applications. Some .ini files, such as **Boot.ini** and **Wininit.ini**, are used by the operating system. The INI File Changes view enables you to specify changes that should be made to .ini files on the target system. Although you can edit any .ini file found on the target system, editing system .ini files is not recommended.



Task ***To edit an .ini file:***

1. Create an .ini file reference.
2. Add a section to the .ini file.
3. Add a keyword to the .ini file.

Before you can create an .ini file reference, you must have at least one component created. If no components exist when your .ini file reference is created, the Create a New Component dialog box is displayed, enabling you to create a component.

Creating .ini File References



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes .ini file functions for modifying .ini file data. You can use these functions in InstallScript projects.

The first step in creating an .ini file change is to create a reference to the file that you would like to edit. In order to do this, you need to know the name and location of the file on the target system that you would like to edit. If the file is not in the location that you specify, the installation cannot make changes to this file.



Task ***To create a reference to an .ini file:***

1. In the View List under **System Configuration**, click **INI File Changes**.
2. Right-click the **INI Files** explorer and then click **Add INI File**.

InstallShield adds a new reference for the .ini file to the INI Files explorer. Configure the .ini file's settings in the right pane. For details about each setting, see .ini File Settings.

After you have created a reference to an .ini file, you can move on to the next step, which is to add a section to your .ini file.

Importing an Existing .ini File



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes .ini file functions for modifying .ini file data. You can use these functions in InstallScript projects.

InstallShield lets you import an existing .ini file into your project. Once you have imported the .ini file, you can use the INI File Changes view to configure the changes that you want to be made on the target system as needed.



Task ***To import an existing .ini file:***

1. In the View List under **System Configuration**, click **INI File Changes**.
2. Right-click the **INI Files** explorer and then click **Import INI File**. The **Open** dialog box opens.
3. Select the .ini file that you want to import, and then click **Open**.

InstallShield imports the .ini file, along with all of its sections, keywords, and values, into your project.

Specifying a Section in an .ini File



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes .ini file functions for modifying .ini file data. You can use these functions in InstallScript projects.

Once you have specified the .ini file that you would like to edit, you can move onto the second step: specifying which section of that file you want to change. The .ini files are divided into sections, with each section containing keywords. Sections are identified by the square brackets surrounding them—for example, **[SectionName]**.



Task ***To specify an .ini file section:***

1. In the View List under **System Configuration**, click **INI File Changes**.
2. Create a reference to an .ini file if it does not exist.
3. In the **INI Files** explorer, right-click the .ini file that should contain the section and click **Add Section**.

InstallShield adds a section to the INI Files explorer. Configure the section's settings in the right pane. For details about each setting, see Section Settings for an .ini File.

After you have specified a section in your .ini file, you can add a keyword.

Specifying a Keyword in an .ini File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes .ini file functions for modifying .ini file data. You can use these functions in InstallScript projects.

The keywords in an .ini file are the lowest level of organization in the .ini file. Keywords store data that must persist between uses of an application.

Once you have added an .ini file to your project and set up one or more sections, you can add keywords to the sections and then configure the keyword's properties. The properties of a keyword include the value for the keyword, as well as the action that should be performed (such as replace a data value or append to an existing data value).



Task ***To specify a keyword for an .ini file:***

1. In the View List under **System Configuration**, click **INI File Changes**.
2. In the **INI Files** explorer, right-click a section and click **Add Keyword**.

InstallShield adds a keyword to the INI Files explorer. Configure the keyword's settings in the right pane. For details about each setting, see Keyword Settings for an .ini File.

Reading Data from .ini Files

You can use the **GetProfString** function to read data (such as key names) from an .ini file, regardless of where the .ini file is located (for example, on a network). The following example script provides a guide:

```
/*

Assuming the .ini file is called Test.ini, and contains the following:

[ProductSettings]
Key1=One
Key2=2
Key3=III
Key4=....

[OtherSettings]
Key1=Value1
Key2=Value2

*/

function OnBegin( )
    STRING svKey1Value; // filled in by GetProfString
begin

// read a single value
GetProfString(
    "\\\\Server\\Config\\Test.ini", // file name; note the double
backslash for each "real" backslash
    "ProductSettings", // section name without square brackets
    "Key1", // key name
    svKey1Value); // STRING variable to capture key value

MessageBox("Key1's value is: " + svKey1Value, INFORMATION);

end;
```

Reading All Key Names from .ini Files

To read all of the key names from a section in an .ini file, use the following script:

```
function OnBegin( )
    STRING svAllKeyNames; // filled in by GetProfString
    LIST listKeyNames; // string list containing key names
begin

// read a single value
GetProfString(
    "\\\\Server\\Config\\Test.ini", // file name; note the double
backslash for each "real" backslash
    "ProductSettings", // section name without square brackets
    "", // key name; null string "" means to get all key names
    svAllKeyNames); // STRING variable to store key names

listKeyNames = ListCreate(STRINGLIST);

// take apart string containing key names
```

```
StrGetTokens(listKeyNames, svAllKeyNames, "");

SdShowInfoList("Key names", "Here they are:", listKeyNames);

// if desired, you can loop over the key names in the list and read each
key's value...

ListDestroy(listKeyNames);

end;
```

Configuring ODBC Resources



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

One of the more complex areas of system configuration involves setting up ODBC drivers, data source names (DSNs), and translators. The ODBC resource must be properly registered on the system with all of the required attributes and, in the case of drivers and translators, install the necessary files, including any installation .dll files. This process is simplified in the ODBC Resources view, in which you can select the drivers, data sources, and translators installed on your development system.

Installing ODBC Resources



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

All of the ODBC drivers, data source names (DSNs), and translators registered on your system are displayed in the ODBC Resources view. DSNs are shown as children of their associated driver. Expand the explorer to view all of the existing ODBC resources.

To include an ODBC resource in your installation, select the check box next to its name in the ODBC Resources pane in the upper-left corner of the view. In installation projects, the resource must then be associated with at least one feature so that it can be installed. Then, you can set the attributes for the selected resource.

Including Additional ODBC Resources



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You can also install ODBC resources not listed in the explorer. Where you add them depends on the type of resource.



Task ***To add a driver that is not listed in the ODBC Resources explorer in the ODBC Resources view:***

1. In the **ODBC Resources** explorer, right-click **Drivers and DSNs** and click **New Driver**.
2. Enter a new name for the driver, or press the F2 key later to rename it.
3. Select the check box next to the new driver to include it in your installation.



Task ***To add a data source that is not listed in the ODBC Resources explorer in the ODBC Resources view:***

1. In the **ODBC Resources** explorer, right-click a driver and click **New DSN**.
2. Enter a new name for the DSN, or press the F2 key later to rename it.
3. Select the check box next to the new DSN to include it in your installation.



***Note** • You cannot add a translator to the list. Instead, you must install it on the development system and then select it.*

Associating ODBC Resources with Features



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Like most of the data in your project, ODBC resources must be associated with a feature. When the feature is installed to the target system, the ODBC resource is installed as a part of the feature.

After you choose to install a driver, DSN, or translator for installation, the Features list is enabled on the right side of the ODBC Resources explorer. Select all of the features to which the ODBC resource belongs. InstallShield creates a new component and associates it with each selected feature. The resource will be installed only once even if it is associated with multiple features, but the resource cannot be installed if none of its features is installed.

In an InstallScript MSI project, if no other feature exists, the resource is added to the DefaultFeature. In a Basic MSI project, if a feature does not exist when you add an ODBC resource to your installation, the Create a New Feature dialog box is displayed. This dialog box prompts you to create a feature. If only one feature is associated with an ODBC resource, it cannot be deselected until you select at least one additional feature.

Setting ODBC Resource Attributes

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When an ODBC resource is registered on the development system, its current attributes are displayed in the Properties list of the ODBC Resources view. You can edit the attributes in the Properties pane after you have selected the driver, data source name (DSN), or translator.

Adding and Deleting New Attribute Entries

Note that adding attributes to a translator is not supported.

Task ***To add a new attribute to a driver or DSN:***

1. Open the **ODBC Resources** view.
2. In the **ODBC Resources** explorer, click the driver or DSN that should contain the new attribute entry.
3. Click in the last row of the property sheet, and add the appropriate information for the **Property** and **Value** columns.

Task ***To delete an attribute from a driver or DSN:***

1. Open the **ODBC Resources** view.
2. In the **ODBC Resources** explorer, click the driver or DSN that should contains the attribute entry that you want to delete.
3. In the property sheet, right-click the row that you want to delete and then click **Delete**.

Using Environment Variables

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The InstallScript language includes the GetEnvVar function for retrieving the current value of an environment variable, and it enables you to create an environment variable.

Environment variables are name and value pairs that can be set on the target system with your installation and can be accessed by your application and by other running programs. Environment variables are stored in the registry.

In the Environment Variables view, you can create, set (or modify), and remove environment variables on the target system through your installation. You can also specify environment variable properties in this view.

Setting Environment Variables

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The InstallScript language includes the GetEnvVar function for retrieving the current value of an environment variable, and it enables you to create an environment variable.

Task ***To create a new environment variable or modify the value of an existing environment variable:***

1. In the View List under **System Configuration**, click **Environment Variables**.
2. Right-click the **Environment Variables** explorer and click **Add Environment Variable**.
3. InstallShield adds a new environment variable with the default name NewEnvironmentVariable*X* (where *X* is a number). Type the name of the variable that you want to modify, remove, or create.
4. In the pane on the right, configure the environment variable's settings. For details about each setting, see Environment Variable Settings.

Environment Variables Example

```
/***************************************************************\
*  The following code creates an environment variable under Windows
*  for an entire system. You can modify the OnEnd event handler
*  function block (or any other function block) to include this example
*  code.
*
*  NOTE: The current user must have administrator privileges for this
*  code to work.
\***************************************************************/

  #define WM_WININICHANGE 0x001A
  #define HWND_BROADCAST 0xffff
  NUMBER nResult;
  STRING szKey, szEnv;
  WPOINTER pEnv;

  begin

      szKey = "SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment";
      RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);
      nResult = RegDBSetKeyValueEx(szKey, "Fame", REGDB_STRING, "C:\\Test", -1);
      if (nResult < 0) then
          MessageBox("Failed to Set Environment Variable", WARNING);
      else
          MessageBox("Successfully Set Environment Variable", INFORMATION);
          // Flush the registry to all applications.
          szEnv = "Environment";
          pEnv = &szEnv;
          SendMessage (HWND_BROADCAST, WM_WININICHANGE, 0, pEnv );
      endif;
      // RebootDialog("", "", SYS_BOOTMACHINE);
  end;


/***************************************************************\
*  The following code creates an environment variable under Windows
*  for the current user. You can modify the OnEnd event handler
*  function block (or any other function block) to include this example
*  code.
*
*  NOTE: The current user must have administrator privileges for this
```

```
*  code to work.
\************************************************************************/

  #define WM_WININICHANGE 0x001A
  #define HWND_BROADCAST 0xffff

  NUMBER nResult;
  STRING szKey, szEnv;
  WPOINTER pEnv;

  begin
      szKey="Environment";
      RegDBSetDefaultRoot(HKEY_CURRENT_USER);
      nResult=RegDBSetKeyValueEx(szKey,"Fame",REGDB_STRING,"C:\\test",-1);
      if (nResult < 0) then
          MessageBox("Failed to Set Environment Variable",WARNING);
      else
          MessageBox("Successfully Set Environment Variable",INFORMATION);
          // Flush the registry to all applications.
          szEnv = "Environment";
          pEnv = &szEnv;
          SendMessage (HWND_BROADCAST, WM_WININICHANGE, 0, pEnv );
      endif;
      //RebootDialog("","",SYS_BOOTMACHINE);
  end;
```

Modifying XML Files



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

You may need to modify .xml files that store settings related to your product, or you may need to modify standard configuration files such as **web.config** and **machine.config**. InstallShield includes support for modifying XML files on the target system at run time. The XML files can be part of your installation, or they can be files that are already present on target systems.

For instructions on how to modify an XML file during installation, consult this section of the documentation.

For background information about XML, see the following Web sites:

- World Wide Web Consortium (http://www.w3.org)
- W3 Schools (http://w3schools.com)

Tip *• Support for XML files extends into other areas of InstallShield product functionality. The System Search feature in InstallShield enables you to search for an attribute value, contents, or existence of the element in the XML file that you specify. For more information, see Searching for XML Data.*

Overview of XML File Changes

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

Design-Time Tasks

The XML File Changes view enables you to define modifications that should be made to an XML file at run time. The typical process for defining those modifications is as follows:

1. If the XML file that you are modifying is part of your installation, add the base XML file to a component in your project through the Files and Folders view.
2. In the XML File Changes view, create an XML file reference for the file that you want to modify.
3. Specify the location for the XML file on the target machine.
4. Specify which feature or features should contain the XML file changes. This may be the same feature that contains the component in step 1, or it may be a different feature.
5. Configure the changes that you want to occur at run time.

You may need to add an MSXML redistributable to your project. For more information, see Run-Time Requirements for XML File Changes.

Run-Time Behavior

The XML File Changes view enables you to configure run-time behavior such as the following:

- Add a namespace mapping to an XML file, and add namespace prefixes to elements and attributes.
- Create a specified element if it does not already exist.
- Modify an element if it exists, but do not create it if it does not exist.
- Modify only the first element in the XML file that matches the specified XPath expression, or modify all matching instances.
- Remove an element during uninstallation.

- Set content for an element.
- Create a new attribute during installation, during uninstallation, or both.
- Remove an existing attribute during installation, during uninstallation, or both.
- Append a string to an existing attribute value during installation, during uninstallation, or both.
- For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, and Transform projects: Replace Windows Installer properties in elements, attributes, attribute values, and element content with the appropriate values at run time.
- For InstallScript projects: Substitute InstallScript string variables for elements, attributes, attribute values, and element content with the appropriate values at run time.

When an installation that contains XML file changes runs on a target system, MSXML parses the XML file and executes the XPath expressions that are associated with the changes that you configured. The Advanced tab in the XML File Changes view for an XML file shows the XPath expressions that are executed on target systems.

When MSXML finds an area of the XML file that matches the XPath expression, the changes that were configured in the XML File Changes view are made.

For examples of how to create some basic XPath expressions, see Using XPath Expressions to Find XML Data in an XML File.



Note *• If the XML file does not exist on the target system at run time and the* ***Always create XML file if it does not already exist*** *check box is selected on the Advanced tab for the XML file, the XML file is created with the XML data that is configured in the XML File Changes view.*

Run-Time Requirements for XML File Changes



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The run-time functionality for XML file changes requires the presence of Microsoft XML Core Services (MSXML) 3.0 or later on the target system.

When an installation that contains XML file changes runs on a target system, MSXML parses the XML file and executes the XPath expressions that are associated with the changes that you configured. When MSXML finds an area of the XML file that matches the XPath expression, the changes that were configured in the XML File Changes view are made.

InstallShield includes redistributables for different versions of MSXML. If it is possible that target systems may not have MSXML, or they may not have the appropriate version of MSXML, you can add the appropriate MSXML redistributable to your project in either the Redistributables view (for Basic MSI and InstallScript MSI projects) or the Prerequisites view (for InstallScript projects).

For detailed information about MSXML, see the MSDN Library.

Creating an XML File Reference



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The recommended way to add an XML file that you want to modify at run time is to add the base XML file to a component in your project through the Files and Folders view. Then add an XML file reference to the XML File Changes view, where you can specify the changes that should be made to that file at run time. The easiest way to add the XML file reference is to import the file into the XML File Changes view.



Important • *The XML File Changes view is not designed to list a node for every node in an XML file. To improve performance, the XML File Changes view should show only the settings that differ from the base XML file:*

- *If the XML file that you are modifying is part of your installation, the XML File Changes view should list only the nodes and node sets that should be added, changed, or deleted after the XML file is installed at run time.*
- *If the XML file that you are modifying is a file that is already present on the target system, the XML File Changes view should list only the nodes and node sets that need to be added, changed, or deleted at run time.*

Therefore, when you are importing a file, import only the nodes in the XML file that you want to modify at run time.

Note that the XML File Changes view does not enable you to specify the order in which new elements should be listed in the XML file. Therefore, importing only the nodes that you want to modify at run time helps to avoid issues that may occur if your product requires that the elements be listed in a particular order.



Task ***To import an XML file into the XML File Changes view:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. Right-click the **XML Files** explorer and then click **Import**. The **Import XML Settings Wizard** opens.
3. Complete the panels in the wizard, selecting only the XML file nodes that you want to change.
4. Click **Import**.

InstallShield adds a new node for the XML file that you imported. The XML file node contains each of the nodes that you selected in the wizard. Each node represents an XPath query that occurs at run time. InstallShield also adds a new component for the XML file that you have imported through the XML File Changes view.

Now you can configure the XML file's settings and specify any element, attribute, and content changes for it.



Tip • *You can also add an XML file by right-clicking the XML Files explorer and then clicking New File. However, in most cases, you may want to create your XML file in a third-party XML editor or text editor. Next, add this file to your project through the Files and Folders view. Then, import the file, as described in the aforementioned procedure, and configure the changes that should be made to the file at run time.*

Specifying the Location of the XML File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

When you add an XML file reference in the XML File Changes view, you need to specify the location of the XML file. If the XML file is part of your installation, the location should match the destination that you specified for the base XML file when you added it to your project through the Files and Folders view.



Task ***To specify the location of the XML file on the target system:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, click the XML file whose location you want to specify.
3. Click the **General** tab.
4. In the **XML File Destination** area, click the **Browse** button. The **Browse for Directory** dialog box opens.
5. In the **Destination Directories** box, select the location.
6. Click **OK**.

InstallShield adds the destination that you specified to the **Specify the location of the XML file on the target machine** box.

Associating an XML File Change Reference with a Feature

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

When you add an XML file reference in the XML File Changes view, InstallShield automatically creates a new component for it. In addition, InstallShield associates the component with the feature that is at the top of your feature list by default. To change this association, you can use the XML File Changes view.

Note *• If your project does not contain any features when you add an XML file reference, InstallShield displays the Create a New Feature dialog box, which lets you to create a new feature.*

Task ***To associate an XML file change reference with a feature in your project:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, click the file whose component you want to associate with a feature.
3. Click the **General** tab.
4. In the **Select Features the XML file belongs to** box, select the check box of any feature that should contain the selected XML file change reference. This may be the same feature that contains the base XML file that you added to your project through the Files and Folders view, or it may be a different feature.

If an end user chooses to install the feature that contains the XML file changes, the XML file changes are performed at run time.

Adding a Root Element

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

If you are adding an XML file reference in the XML File Changes view, you can add a root element. Note that an XML file can contain only one root element.

Task ***To add a root element to an XML file:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, right-click the XML file to which you want to add a new root element and click **New Root Element**.

InstallShield adds the new root element. Use the tabs that are displayed in the right pane to configure its settings.

Tip *• To add a root element and one or more levels of subelements in one step, type the name of the root element plus the subelements, with each separated by a slash. For example, to add a root element called* ***Root****, a* ***Sub*** *element under the* ***Root*** *element, and a* ***Sub2*** *element under the* ***Sub*** *element, type the following:*

Root/Sub/Sub2

InstallShield automatically expands it in the explorer.

Adding a New Element

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

You can add a new element to the root element or any element in your XML file.

Task ***To add a new element:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, right-click the XML file to which you want to add a new element and click **New Element**.

InstallShield adds the new element. Rename it as appropriate, and use the tabs that are displayed in the right pane to configure its settings.

Tip • *When you are adding elements and subelements under the root element in the XML File Changes view, you can type the full XPath expression for the name of the element, and it will automatically expand in the explorer.*

To add a root element and one or more levels of subelements in one step, type the name of the root element plus the subelements, with each separated by a slash. For example, to add a root element called ***Root****, a* ***Sub*** *element under the* ***Root*** *element, and a* ***Sub2*** *element under the* ***Sub*** *element, type the following:*

Root/Sub/Sub2

InstallShield automatically expands it in the explorer so that each element has its own node.

Adding an Element for a .NET Configuration File

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

If you define .NET configuration settings for your product in a web.config file, you can specify those settings through the XML File Changes view.

Task ***To add an element for a .NET configuration file:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, right-click the XML file to which you want to add a configuration element, click **Add Predefined Element**, point to **.NET Configuration Files**, point to **Web Configuration File**, point to a given set of settings, and click the appropriate command.

InstallShield adds the new element in the explorer.

For detailed reference information about the different elements and attributes in .NET configuration files, see Configuration File Schema on the MSDN Web site (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/gngrfnetframeworkconfigurationfileschema.asp).

Adding an Attribute to an Element

Project • *This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

You can add attributes to an element in your XML file and then specify whether the attribute should be created at run time, the attribute should be removed at run time, or the attribute value should be appended to the existing value at run time. You can also schedule whether the task should occur during installation, uninstallation, or both.

If an attribute that you add already exists in the XML file on the target machine but the attribute has a different value than the one that you specified in the XML File Changes view, the installation updates the value at run time.

Task ***To add an attribute to an XML element:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, click the XML file to which you want to add an attribute.
3. Click the **General** tab.
4. Click the **Add** button.

InstallShield adds a new row—with default values for the Attribute, Value, Operation, and Scheduling columns—to the attribute table. Change any of the settings as needed. To learn more about each of the columns, see General Tab for an XML Element.

Tip *• If you configure your installation so that the XML file remains on the target system when its component is uninstalled, you may want to create installation/uninstallation attribute pairs in the attribute table.*

*For example, to add a **key** attribute during installation and remove that same **key** attribute during uninstallation, add two rows with the **key** attribute to the attribute table; schedule one row for installation, and the other for uninstallation.*

Editing an Attribute's Value

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

InstallShield enables you to edit the value of an attribute if it is already present on the target machine at run time. To do so, add the attribute to the element in the XML File Changes view, and configure the value for the attribute as needed. The procedure for editing an attribute value is the same as adding an attribute value. To learn more, see Adding an Attribute to an Element.

At run time, if a different value exists for the attribute, the installation replaces the value with the one that you configured in the XML File Changes view.

Adding Content to an Element



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

XML elements can contain text content. For example, in the following XML code, **feature** is the content for the element **product**:

```
<product>feature</product>
```



Task ***To add content to one of the elements listed in the XML File Changes view:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, select the element to which you want to add content.
3. Select the **Advanced** tab.
4. Select the **Set element content** check box.
5. In the **Content** box, type the text content.

   When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) next to this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Using XPath Expressions to Find XML Data in an XML File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

In order for your installation to add a new element or attribute to an XML file, or to perform some other change to an XML file at run time, MSXML must use the XPath expressions that are defined in the XML File Changes view to navigate through the XML file and locate the areas that need to be modified.

For detailed information about writing XPath expressions, see the following Web sites:

- World Wide Web Consortium (http://www.w3.org)
- W3 Schools (http://w3schools.com)

The following sections show examples of how to create some basic XPath expressions.

Example 1: Adding Two Elements

In example 1, the installation adds the bold lines in the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Books>
   <Biographies>
      <Book Author="Bill Smith" Copyright="2007">Bill's Great Biography</Book>
      <Book Author="John Smith" Copyright="2006">John's Great Biography</Book>
   </Biographies>
</Books>
```

To add those elements, the following XPath expressions are added under the Biographies node in the XML File Changes view:

```
Book[@Author="John Smith"]
Book[@Author="Bill Smith"]
```

The following screen shots show the settings in the XML File Changes view.

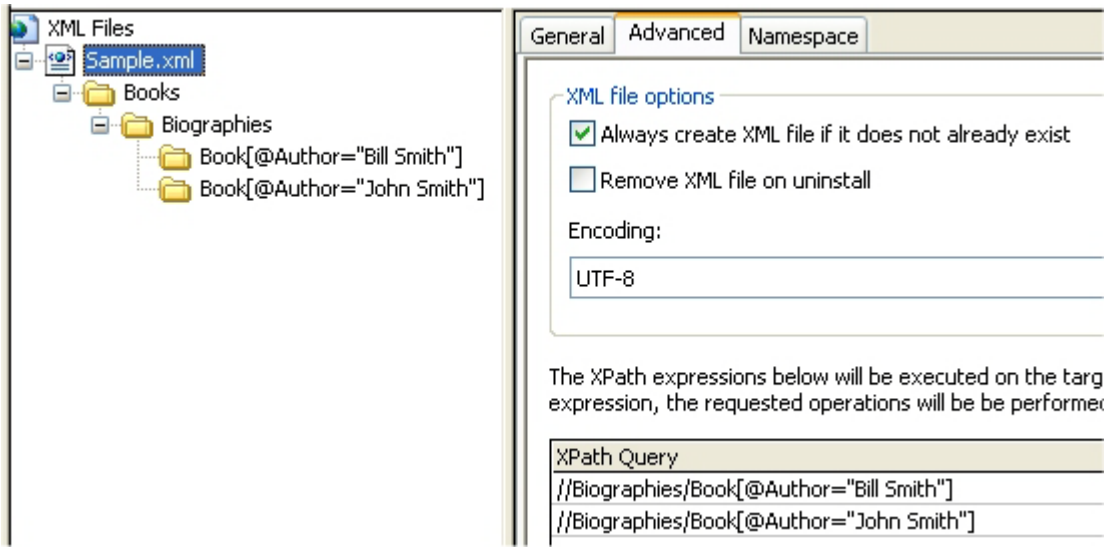


Figure 4-1: Settings for the Sample.xml File

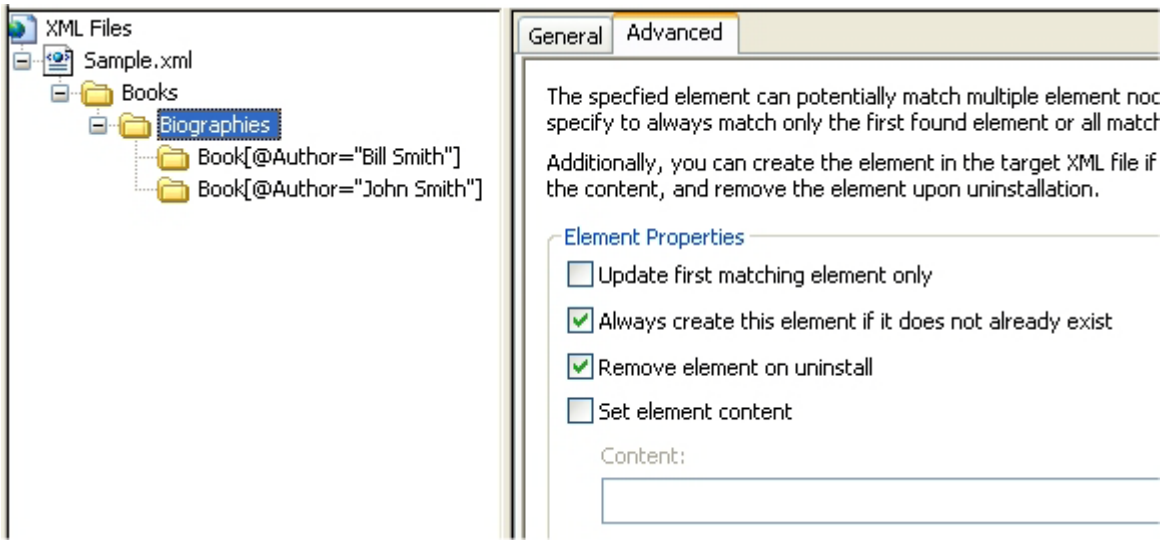


Figure 4-2: Settings for the Biographies Node, which Contains the Child Nodes to Be Added

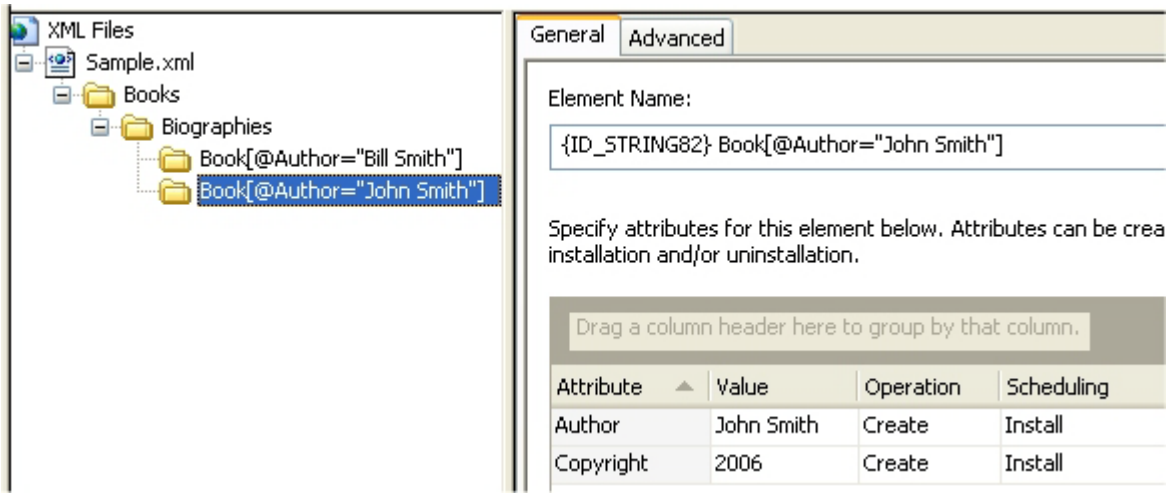


Figure 4-3: General Settings for One of the Child Nodes to Be Added

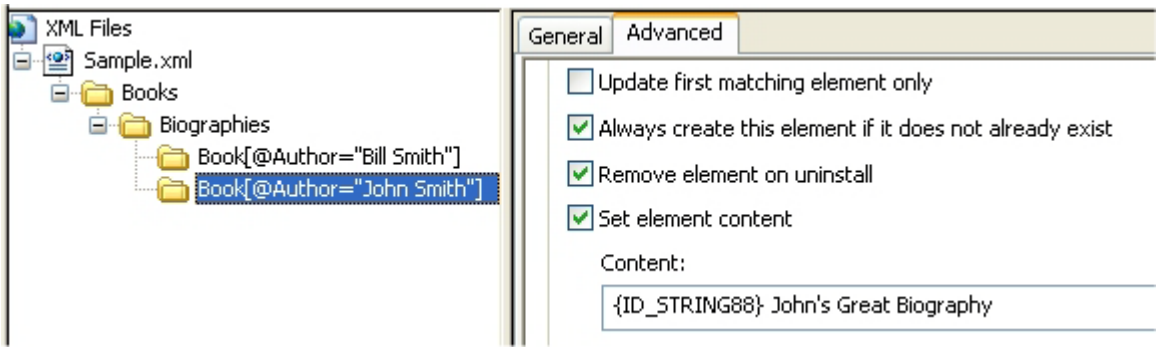


Figure 4-4: Advanced Settings for One of the Child Nodes to Be Added

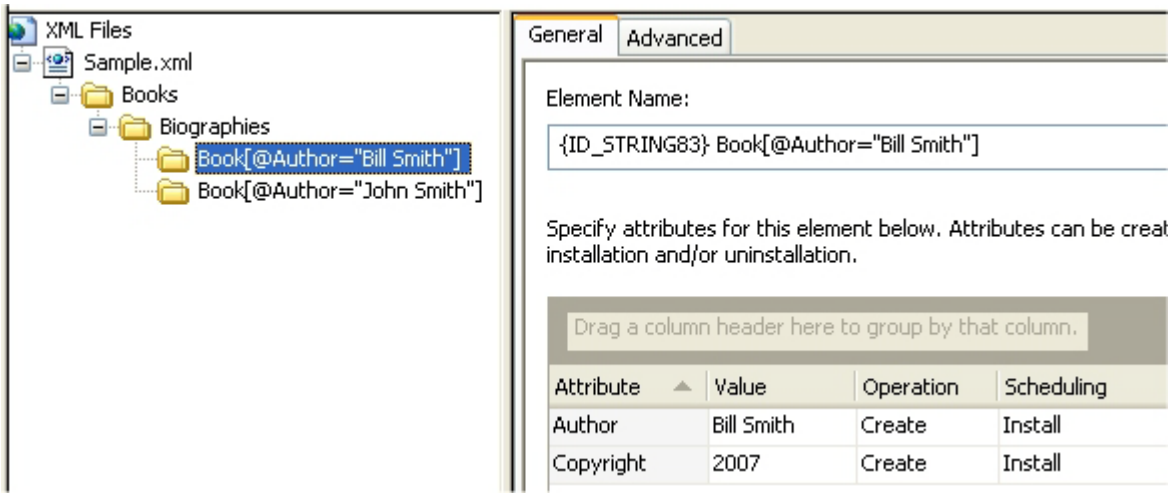


Figure 4-5: General Settings for One of the Child Nodes to Be Added

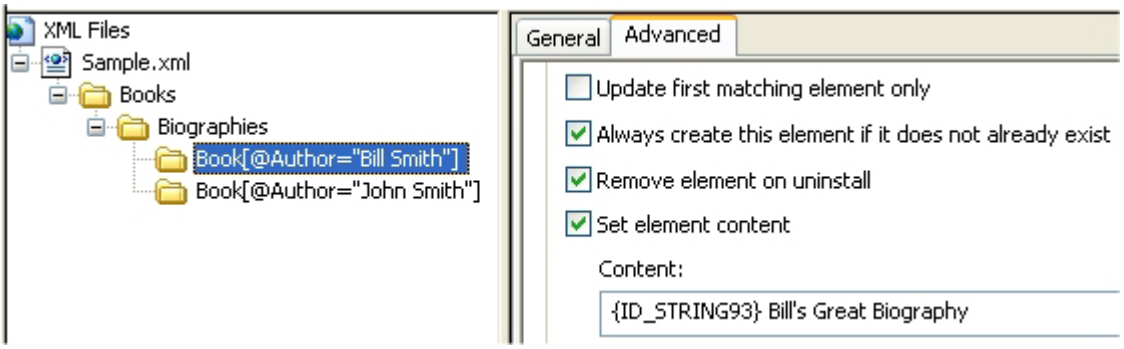


Figure 4-6: Advanced Settings for One of the Child Nodes to Be Added

Example 2: Adding Attributes to Elements

In example 2, the installation adds the bold attributes and values in the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Books>
   <Biographies>
      <Book Author="Bill Smith" Copyright="2007" Publisher="Bill & John's Publish Co.">Bill's Great
Biography</Book>
      <Book Author="John Smith" Copyright="2006" Publisher="Bill & John's Publish Co.">John's Great
Biography</Book>
   </Biographies>
</Books>
```

The following screen shot shows the settings in the XML File Changes view.

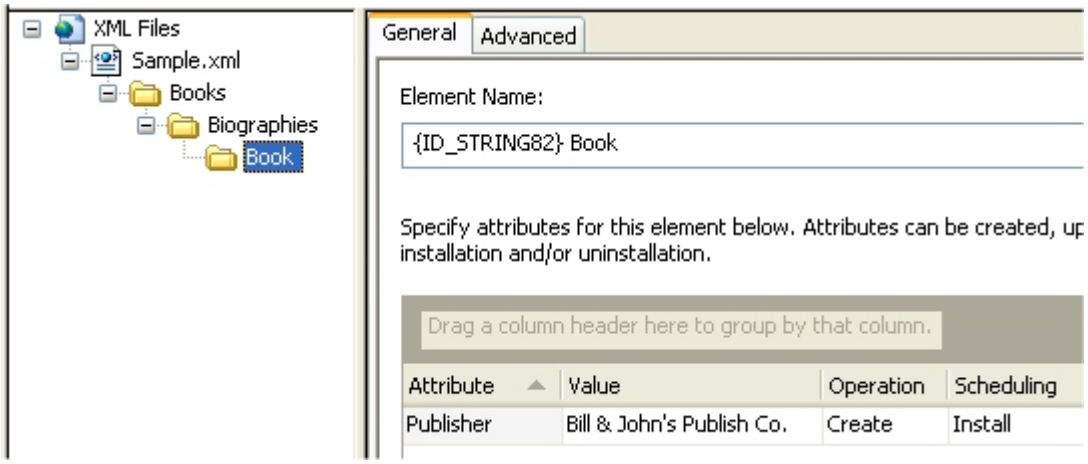


Figure 4-7: Settings for the Book Node

Example 3: Updating an Attribute's Value

In example 3, the installation searches for the attribute that has the 2006 Copyright value (as shown in bold in the following XML document) and replaces its value with a 2007 Copyright value:

```
<?xml version="1.0" encoding="UTF-8"?>
<Books>
   <Biographies>
      <Book Author="Bill Smith" Copyright="2007" Publisher="Bill & John's Publish Co.">Bill's Great
Biography</Book>
      <Book Author="John Smith" Copyright="2006" Publisher="Bill & John's Publish Co.">John's Great
Biography</Book>
   </Biographies>
</Books>
```

To update the value, the following XPath expression is added under the Biographies node in the XML File Changes view:

```
Book[@Copyright="2006"]
```

The following screen shot shows the settings in the XML File Changes view.

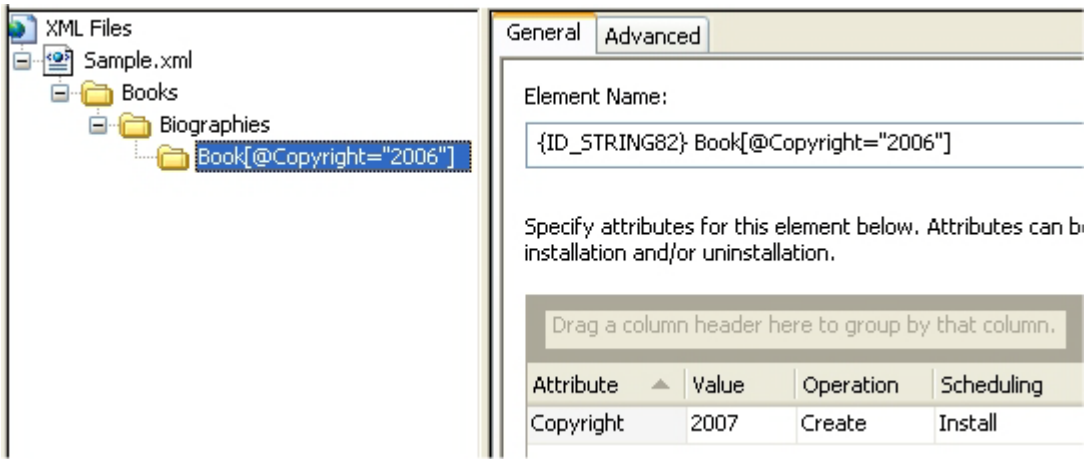


Figure 4-8: General Settings for the Attribute to Be Updated

Using Windows Installer Properties to Dynamically Modify XML Files



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, and Transform projects, you can use a Windows Installer property to specify an XML element, an attribute, an attribute value, or an element's content. At run time, Windows Installer uses **MsiFormatRecord** to resolve the property value, and it uses that value as the data in your XML file. This enables you to use data that end users enter in dialogs, or other configuration information that is determined during the installation, in your product's XML files.

For example, your installation may include the SQL Login dialog, which lets end users select a SQL Server. The name of the server that they select is typically stored in the **IS_SQLSERVER_SERVER** property. If you want an XML file to contain the name of the SQL server that an end user selects, you can use this property, surrounded by brackets (that is, **[IS_SQLSERVER_SERVER]**), when you configure your XML file changes in the XML File Changes view. Then at run time, Windows Installer automatically replaces the property with its associated value.



Tip • *Use a Windows Installer property for an element, an attribute, or element content only if the value of the property would make a well-formed XML document. If a property value would lead to syntax errors in the XML document, your product or installation may not behave as expected.*

*For example, if you set the name of an element to **[INSTALLDIR]**—a Windows Installer property whose value is a path on the target system—the result at run time is an element name that contains backslashes. Element names cannot contain backslashes, and XML documents that contain element names with backslashes are not valid.*



Note • *If test your XML file from within the XML File Changes view, any Windows Installer properties that are used for XML data are not replaced with the appropriate values during testing. For more information about testing, see Testing Installation Changes to an XML File or Testing Uninstallation Changes to an XML File.*

Example

The following procedure demonstrates how to use the name of the SQL Server that an end user selects in the SQL Login dialog as the content for one of the elements in your XML file. Note that you can substitute a hard-coded value with a property for any of the elements, attributes, attribute values, or element content in the XML File Changes view. The property that you specify in this view must be enclosed within square brackets, and the property name must be in all uppercase letters; for example, **[MYPROPERTY]**.

Task ***To use the name of the SQL Server that end users select as the content for an XML element:***

1. Configure the SQL connection and any associated SQL scripts if you have not already done so, and determine the name of the server property name:
   a. In the View List under **Server Configuration**, click **SQL Scripts**.
   b. Add a SQL connection and SQL scripts as needed. For instructions, see Configuring SQL Support.
   c. In the **SQL Scripts** explorer, click the SQL connection.
   d. Click the **Advanced** tab.
   e. Note the name that is selected for the **Target Server Property Name** setting. The default value is typically **IS_SQLSERVER_SERVER**.
2. In the View List under **System Configuration**, click **XML File Changes**.
3. In the **XML Files** explorer, select the element to which you want to add content.
4. Select the **Advanced** tab.
5. Select the **Set element content** check box.
6. In the **Content** box, type the name that is selected in step 1e, and enclose it within brackets. For example:

   [IS_SQLSERVER_SERVER]
7. Build your release.

Using InstallScript Text Substitution to Dynamically Modify XML Files

Project *• This information applies to InstallScript projects.*

For InstallScript projects, you can use a text substitution string variable for an XML element, an attribute, an attribute value, or an element's content. When the XML file changes occur at run time, the InstallScript run-time code uses the **TextSubSubstitute** function to replace the string variable with the appropriate value. This enables you to use data that end users enter in dialogs, or other configuration information that is determined during the installation, in your product's XML files.

Tip *• Use text substitution for an element, an attribute, or element content only if the value of the property would make a well-formed XML document. If a property value would lead to syntax errors in the XML document, your product or installation may not behave as expected.*

For example, if you set the name of an element to a text substitution string variable that will be replaced with text that includes one or more spaces, end users may have problems with your product or your installation. Element names cannot contain spaces, and XML documents that contain element names with spaces are not valid.

Note • *If test your XML file from within the XML File Changes view, any text substitution string variables that are used for XML data are not replaced with the appropriate values during testing. For more information about testing, see Testing Installation Changes to an XML File or Testing Uninstallation Changes to an XML File.*

Example

The following procedure demonstrates how to use the name of the SQL Server that an end user selects in the SQL Login dialog as the content for one of the elements in your XML file. Note that you can substitute a hard-coded value with a text substitution string variable for any of the elements, attributes, attribute values, or element content in the XML File Changes view. The string variable that you specify in this view must be enclosed within angle brackets; for example, **<MYPROPERTY>**. The string variable that you specify is case-sensitive.

Task ***To use the name of the SQL Server that end users select as the content for an XML element:***

1. Configure the SQL connection and any associated SQL scripts if you have not already done so:
   a. In the View List under **Server Configuration**, click **SQL Scripts**.
   b. Add a SQL connection and SQL scripts as needed. For instructions, see Configuring SQL Support.
2. In the View List under **System Configuration**, click **XML File Changes**.
3. In the **XML Files** explorer, select the element to which you want to add content.
4. Select the **Advanced** tab.
5. Select the **Set element content** check box.
6. In the **Content** box, type the following:

   <MYPROPERTY>
7. In the View List under **Behavior and Logic**, click **InstallScript**.
8. Find the dialog code in the OnSQLLogin event for the SQL Login dialog that contains the SQL Server control, and add a call to the InstallScript function **TextSubSetValue**. For example:

```
// Display login dialog (without connection name)
// COMMENT OUT TO SWAP DIALOGS
nResult = SQLServerSelectLogin2( szConnection, szServer, szUser, szPassword, bWinLogin, szDB, FALSE,
TRUE );
TextSubSetValue("<MYPROPERTY>", szServer, TRUE);
```

9. Build your release.

Using Reserved Characters (<, >, &, ', and ") Inside Elements

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

If you use a reserved character such as a less than symbol (<) inside an XML element, the MSXML parser converts it at run time on target systems to its predefined entity (<). Using the less than symbol instead of its entity would generate an error because XML parsers would interpret it as the start of a new element, and it would result in invalid XML code. Note that if you open the resulting XML file in a browser such as Internet Explorer, the character—not its entity—is displayed inside the XML element.

The following table lists the reserved XML characters and their entity equivalents. The table also indicates whether each reserved character is replaced by its entity at run time.

Table 4-8 • Reserved Characters and Their Predefined Entities

Character	Entity	Description	Notes
<	<	Less than	This character cannot be used as content in an XML element because it is reserved to be used to indicate the start of an XML element. This character is automatically replaced by its entity at run time.
>	>	Greater than	This character is automatically replaced by its entity at run time.
&	&	And	This character cannot be used as content in an XML element unless it indicates the start of an entity. If this character is used as content in an XML element but it does not indicate the start of an entity, it is automatically replaced by its entity at run time.
'	'	Apostrophe	This character is not automatically replaced by its entity at run time.
"	"	Quotation mark	This character is not automatically replaced by its entity at run time.

Using Namespaces in XML Files



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

XML namespaces provide a method for avoiding element name conflicts. When you are specifying XML file changes in InstallShield, you can specify the namespace mappings that should be declared in the XML file and then specify namespace prefixes for any of the file's elements.



Tip • *If you use the Import XML Settings Wizard to import an XML file into the XML File Changes view, InstallShield imports any namespaces that are declared for the file.*

Declaring Namespace Mappings for an XML File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

You can declare namespace mappings for an XML file in the XML File Changes view.



Task ***To declare a namespace mapping for an XML file:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, click the XML file that should contain the namespace.
3. Click the **Namespace** tab.
4. Click a row in the table to add a new namespace.
5. In the **Prefix** column, type the prefix that should be used for any elements that are associated with the corresponding namespace.
6. In the **URI** column, type a URL or a string of characters that identifies the Internet resource.

Adding a Namespace Prefix to an Element



Project • *This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

If you have declared a namespace mapping for an XML file, you can associate any element in the file with that namespace by adding the corresponding prefix to the element.



Task ***To add a namespace prefix to an element:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, right-click the element to which you want to add a prefix and then do one of the following:
   - To add a prefix to only the selected element, point to **Namespace Prefix**, and then click to the appropriate namespace mapping.
   - To add a prefix to the element and all of its subelements, point to **Namespace Prefix (include all subelements)**, and then click to the appropriate namespace mapping.

InstallShield adds the prefix to the element (and all of its subelements, if appropriate) in the XML Files explorer.



Tip *• You can also add a namespace prefix by right-clicking an element, clicking Rename, and adding the prefix and the colon (:) before the element name.*

Adding a Namespace Prefix to an Attribute



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

If you have declared a namespace mapping for an XML file, you can associate any attribute in the file with that namespace by adding the corresponding prefix to the attribute.

Task ***To add a namespace prefix to an attribute:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, click the element that contains the attribute.
3. In the right pane, click the **General** tab.
4. In the grid, double-click the attribute to which you want to add the prefix, and then place the cursor at the start of the attribute name.
5. Add the prefix and a colon (:) before the attribute name.

Removing a Namespace Prefix from an Attribute

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

Task ***To remove a namespace prefix from an attribute:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, click the element that contains the attribute.
3. In the right pane, click the **General** tab.
4. In the grid, double-click the attribute from which you want to remove the prefix, and then place the cursor at the start of the attribute name.
5. Delete the prefix and a colon (:) before the attribute name.

Removing a Namespace Prefix from an Element

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

- *Merge Module*
- *MSI Database*
- *Transform*



Task ***To remove a namespace prefix from an element:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, right-click the element whose prefix you want to remove, and then do one of the following:
   - To remove the prefix from only the selected element, point to **Namespace Prefix**, and then click **<None>**.
   - To remove the prefix from the element and all of its subelements, point to **Namespace Prefix (include all subelements)**, and then click **<None>**.

InstallShield removes the prefix from the element (and all of its subelements, if appropriate) in the XML Files explorer.

Removing Namespace Mappings from an XML File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*



Task ***To remove a namespace mapping from an XML file:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, click the XML file that contains the namespace that you want to remove.
3. Click the **Namespace** tab.
4. Click the row that contains the namespace mapping that you want to remove, and then click the **Delete** button.

InstallShield removes the namespace from the table.

Testing Installation Changes to an XML File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

InstallShield enables you to test just the XML file changes that are configured for your project through the XML File Changes view without requiring you to build and run your entire installation. When you test the installation changes, InstallShield uses the latest version of MSXML that you have on your machine to parse the XML file and execute the XPath expressions that you configured. When MSXML finds an area of the XML file that matches the XPath expression, the changes that were configured in the XML File Changes view are made.



***Note** • If your XML file changes include Windows Installer properties or InstallScript text substitutions, they are not replaced with the appropriate values during testing.*



Task ***To test installation changes to an XML file:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, right-click the XML file that you want to test and then click **Test XML File Install Changes**. The **Select Test XML File** dialog box opens.
3. In the **File name** box, select the target file to which you want the installation changes applied, or specify a new target file name and location:
   - If you are modifying an XML file that is installed as part of your installation, select a copy of that file. (Do not select the actual file in your installation because testing the XML installation changes would modify it.)
   - If you are modifying an XML file that already exists on the target system, select a copy of that file.
   - To test what happens if the XML file does not exist at run time and it is not installed as part of your installation, specify a new file name and location.

   The default file name is the name of the test file that you right-clicked in step 2.
4. Click **Open**.

If the target file already exists, InstallShield applies the changes from the test file to the target file. If the target file that you specified does not exist and the **Always create XML file if it does not already exist** check box is selected on the Advanced tab for the XML file, InstallShield creates the file and applies the changes from the test file.

InstallShield displays details about the installation test on the Results tab of the Output window. The details include a hyperlink to the test file.



***Tip** • If the target file is open in a browser window when you perform the testing, you may need to refresh the browser to see the test changes.*

Testing Uninstallation Changes to an XML File

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

After you have tested the XML file installation changes that you have configured through the XML File Changes view, you may want to test the XML file changes that occur during uninstallation. This enables you to determine whether the changes that you configured behave as you expected during uninstallation.

When you test the uninstallation changes, InstallShield uses the latest version of MSXML that you have on your machine to parse the XML file and execute the XPath expressions that you configured. When MSXML finds an area of the XML file that matches the XPath expression, the changes that were configured in the XML File Changes view are made.

Note *• If your XML file changes include Windows Installer properties or InstallScript text substitutions, they are not replaced with the appropriate values during testing.*

Task

To test uninstallation changes to an XML file:

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, right-click the XML file that you want to test and then click **Test XML File Uninstall Changes**. The **Select Test XML File** dialog box opens.
3. In the **File name** box, select the target file to which you want the uninstallation changes applied.

   The default value is the name of the file whose installation changes you last tested.
4. Click **Open**.

InstallShield applies the uninstallation changes that are configured in the XML File Changes view to the test file.

InstallShield displays details about the uninstallation test on the Results tab of the Output window. The details include a hyperlink to the test file. Note that if you have configured the XML file to be removed during uninstallation, the hyperlink may not work, since the file may no longer be present.

Tip *• If the target file is open in a browser window when you perform the testing, you may need to refresh the browser window to see the test changes.*

Removing an Element or an XML File from the XML File Changes View

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

Task ***To remove an element or an XML file from the XML File Changes view:***

1. In the View List under **System Configuration**, click **XML File Changes**.
2. In the **XML Files** explorer, right-click the XML file or XML element that you want to remove and click **Delete**.

If you delete an XML file, InstallShield displays a message that explains that the component associated with the XML file will be deleted along with the file itself.

Modifying Text Files

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals. You can use these functions in InstallScript projects.

InstallShield enables you to configure search-and-replace behavior for content in text files—for example, .txt, .htm, .xml, .config, .ini, and .sql files—that you want to modify at run time on the target system. The text files can be part of your installation, or they can be files that are already present on target systems.

The Text File Changes view is where you define the changes that you want to be made to the text files. This view lets you do the following:

- Add one or more text change sets to your project. A text change set is a reference to one or more text files that you want to search at run time.
- Add one or more text change items to a text change set. A text change item identifies the search-and-replace criteria.

Task ***To configure text file changes:***

1. Create a text change set.
2. Specify the change for the text file.

For more information on how to modify a text file at run time, consult this section of the documentation.

Creating a Text Change Set

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals. You can use these functions in InstallScript projects.

The first step in configuring text file changes is to create a reference to the file (or files) that you would like to edit; this reference is called a *text change set*. The file can be any non-binary file—for example, a .txt, .htm, .xml, .config, .ini, or .sql file. The file can be part of your installation (that is, one that you added to your project in the Files and Folders view), or it can be a file that is already present on target systems.

Note *• Each text change set must be associated with a component in your project. Therefore, before you can create a text change set, your project must have at least one component. If no components exist when you are creating a text change reference, the Create a New Component dialog box is displayed, enabling you to create a component.*

Task ***To create a text change set to reference one or more text files that you want to change at run time:***

1. In the View List under **System Configuration**, click **Text File Changes**.
2. Right-click the **Text File Changes** explorer and then click **Add Change Set**.

   InstallShield adds a new change set with a default name.
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.

   The name is not displayed at run time; it is an internal name that is used to differentiate between various change sets in your project.
4. In the right pane, configure the settings for the change set. For details about each setting, see Change Set Settings.

After you have created a reference to a text file and configured its settings, you can move on to the next step, which is to specify the search-and-replace criteria.



***Tip** • You can use Windows Installer public properties to specify the names of the text files that you want to include in or exclude from your search. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's text files are modified at run time. To learn more, see Using Windows Installer Properties to Dynamically Modify Text Files.*

Specifying a Change for a Text File



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals. You can use these functions in InstallScript projects.

Once you have specified the text file that you would like to edit, you can move onto the next step of configuring text file changes: specifying the search-and-replace criteria. Each set of criteria is known as a *change item*.



Task ***To specify the change that you want to be made for a text file:***

1. In the View List under **System Configuration**, click **Text File Changes**.
2. In the **Text File Changes** explorer, right-click the change set item whose text changes you want to define, and then click **Add Change**.

   InstallShield adds a new change item with a default name.
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.

   The name is not displayed at run time; it is an internal name that is used to differentiate between various change items in your project.
4. In the right pane, configure the settings for the change. For details about each setting, see Change Set Settings.

To modify additional strings in the text file, add additional change items to the change set in this view—one for each string change.



***Tip** • You can use Windows Installer public properties to specify the search strings and the replacement strings. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run*

time, when your product's text files are modified at run time. To learn more, see Using Windows Installer Properties to Dynamically Modify Text Files.

Changing the Order in Which Text File Changes Are Made

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals. You can use these functions in InstallScript projects.

Text file changes are made on the target system in the order in which they are listed in the Text File Changes view.

Task ***To change the order in which text file changes are made at run time:***

1. In the View List under **System Configuration**, click **Text File Changes**.
2. In the **Text File Changes** explorer, right-click one of the change set items or change items that you want to move, and then click either **Move Up** or **Move Down**.

Repeat the last step until all of the text file changes are correctly sorted.

Using Windows Installer Properties to Dynamically Modify Text Files

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals. You can use these functions in InstallScript projects.

You can use a Windows Installer property to specify text strings for which you are searching or modifying. You can also use a property to specify the text files that you are including or excluding in your search.

At run time, Windows Installer uses **MsiFormatRecord** to resolve the property value, and it uses that value to modify your text file. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's text files are modified at run time.

Example

The following procedure demonstrates how to let end users specify during installation an IP address that must be written to an XML-based **web.config** file at run time. The **web.config** file is installed with the product to **INSTALLDIR**, and it contains XML such as the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appSettings>
        <add key="IP Address" value="default" />
    </appSettings>
</configuration>
```

The default value in bold must be replaced by the IP address that an end user enters.

Note that you can substitute a hard-coded value with a property for the following change set settings in the Text File Changes view:

- Include Files
- Exclude Files

In addition, you can use a property for the following change item settings in the Text File Changes view:

- Find What
- Replace With
- Insert Text

The property that you specify in any of these settings must be enclosed within square brackets, and the property name must be all uppercase; for example, **[MYPROPERTY]**.

Step 4 of the procedure is slightly different, depending on the project type, since Windows Installer controls the user interface of Basic MSI installations, and the InstallScript engine controls the user interface of InstallScript MSI installations.



Task ***To let end users specify the IP address:***

1. In the View List under **System Configuration**, click **Text File Changes**.
2. Add and configure a change set item, which identifies the file for which you want the installation to search:
    a. Right-click the **Text File Changes** explorer and then click **Add Change Set**.

       InstallShield adds a new change set item. Steps 2b through 2d explain how to configure its settings, which are displayed in the right pane.

    b. In the **Target Folder** setting, select the **[INSTALLDIR]** directory property.

    c. In the **Include Files** setting, enter the following:

       web.config

d. Leave the default values for the other settings.

3. Add and configure a change item, which identifies the search-and-replace criteria:

a. In the **Text File Changes** explorer, right-click the change set item that you created in step 2, and then click **Add Change**.

InstallShield adds a new change item. Steps 3b through 3e explain how to configure its settings, which are displayed in the right pane.

b. In the **Action** setting, select **Replace**.

c. In the **Find What** setting, enter the following:

<add key="IP Address" value="default"

d. In the **Replace With** setting, enter the following:

<add key="IP Address" value="[MYPROPERTY]"

e. Leave the default values for the other settings.

4. Use the property in a dialog. This part of the procedure depends on which project type you are using.

- For Basic MSI projects:

a. In the View List under **User Interface**, click **Dialogs**.

b. In the **Dialogs** explorer, expand the **All Dialogs** folder, and click the language under the dialog that should contain the User Name control. As an alternative, you can add a new dialog.

c. Add an **Edit Field** control to the dialog, and set its **Property** property to the following:

MYPROPERTY

- For InstallScript MSI projects:

a. In the View List under **Behavior and Logic**, click **InstallScript**.

b. Find the dialog code in the OnFirstUIBefore event for the dialog that should contain the User Name control, and add a call to the Windows Installer API function **MsiSetProperty**. For example, if you want end users to enter the IP address in an edit box on the SdShowDlgEdit1 dialog that you have added to your project, you would add an **MsiSetProperty** call as shown in the following lines of code:

```
Dlg_SdShowDlgEdit1:
    nResult = SdShowDlgEdit1 (szTitle, szMsg, szField1, svEdit1);
    MsiSetProperty (ISMSI_HANDLE, "MYPROPERTY", svEdit1);
    if (nResult = BACK) goto Dlg_SdWelcome;
```

5. Build your release.



Tip • *If you are creating a multilanguage project and you want to use a property that can have different values based on the language that your installation uses, you can use a localizable property. For more information, see Creating a Localizable Property.*

Specifying the Code Page that Should Be Used for Opening ANSI Text Files

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals. You can use these functions in InstallScript projects.

When an installation opens an ANSI text file to make the changes that are configured in the Text File Changes view, the installation uses the code page that is specified in the CodePage column of the **ISSearchReplaceSet** table of your project. The default value of this column is the number 0, which is CP_ACP, the code page that is currently configured to be the system Windows ANSI code page on the target system. You can use the Direct Editor view to override this value with a specific code page.

Task ***To override the default code page that should be used to open an ANSI text file:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, click the **ISSearchReplaceSet** table. InstallShield displays the table in the right pane.
3. In the grid, find the row that corresponds with the change set item whose code page you want to configure.

   The ISSearchReplaceSet column in this grid shows all of the names of the change sets that are available in the **Text File Changes** explorer in the **Text File Changes** view.
4. Change the value of the **CodePage** field in the appropriate row.

Removing a Change Item or a Change Set from the Text File Changes View

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This information does not apply to InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals. You can use these functions in InstallScript projects.

If you no longer want a particular change to be performed to a text file, you can remove its change item from the Text File Changes view. In addition, if you no longer want a file or group of text files to be searched, you can remove the corresponding change sets from the Text File Changes view.

Task ***To remove a change item or a change set from the Text File Changes view:***

1. In the View List under **System Configuration**, click **Text File Changes**.
2. In the **Text File Changes** explorer, right-click the change set item or change item that you want to remove, and then click **Delete**.

InstallShield removes the item from the Text File Changes view.

Scheduling Tasks

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

You can have your installation create and configure automated tasks through the Windows task scheduler at run time on target systems. The scheduled tasks can launch files that are part of your installation, or files that are already present on target systems.

Adding a Scheduled Task

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

Note *• Each scheduled task must be associated with a component in your project. Therefore, before you can add a scheduled task, your project must have at least one component. If no components exist when you are adding a scheduled task, the Create a New Component dialog box is displayed, enabling you to create a component.*



Task ***To add a scheduled task to your project:***

1. In the View List under **System Configuration**, click **Scheduled Tasks**.
2. Right-click the **Scheduled Tasks** explorer and then click **Add Scheduled Task**. InstallShield adds a new task with a default name.
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.

   The name is not displayed at run time; it is an internal name that is used to differentiate between various scheduled tasks in your project.
4. In the right pane, configure the settings for the task. For details about each setting, see Scheduled Tasks Settings.

Using Windows Installer Properties to Dynamically Configure a Scheduled Task



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

You can use a Windows Installer property to specify information such as the account information that should be used to run the file that a scheduled task launches.

At run time, Windows Installer uses **MsiFormatRecord** to resolve the property value, and it uses that value to configure your scheduled task. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's scheduled tasks are created at run time.

Example

The following procedure demonstrates how to let end users specify during installation an account and password that should be used to run the scheduled task. The properties that you specify must be enclosed within square brackets, and the property names must be all uppercase; for example, **[MYPROPERTY]**.

Step 4 of the procedure is slightly different, depending on the project type, since Windows Installer controls the user interface of Basic MSI installations, and the InstallScript engine controls the user interface of InstallScript MSI installations.



Task ***To let end users specify account and password information for the scheduled task:***

1. In the View List under **System Configuration**, click **Scheduled Tasks**.
2. In the **Scheduled Tasks** explorer, select the task that you want to configure.
3. In the **Run As** setting, enter the following:

   [DOMAINNAME]\[USERNAME]
4. In the **Password** setting, enter the following:

   [PASSWORD]
5. Use the properties in a dialog. This part of the procedure depends on which project type you are using.
   - For Basic MSI projects:
     a. In the View List under **User Interface**, click **Dialogs**.
     b. In the **Dialogs** explorer, expand the **All Dialogs** folder, and click the language under the dialog that should contain Domain Name, User Name, and Password controls. As an alternative, you can add a new dialog.
     c. Add an **Edit Field** control to the dialog, and set its **Property** property to the following:

        DOMAINNAME
     d. Repeat step 5c for the **USERNAME** and **PASSWORD** properties.
   - For InstallScript MSI projects:
     a. In the View List under **Behavior and Logic**, click **InstallScript**.
     b. Find the dialog code in the OnFirstUIBefore event for the dialog that should contain the Domain Name, User Name, and Password controls, and add a call to the Windows Installer API function **MsiSetProperty**. For example, if you want end users to enter the information in edit boxes on the **SdShowDlgEdit3** dialog that you have added to your project, you would add **MsiSetProperty** calls as shown in the following lines of code:

```
Dlg_SdShowDlgEdit3:
    nResult = SdShowDlgEdit3
        (szTitle, szMsg, szField1, szField2, szField3, svEdit1, svEdit2, svEdit3);
    MsiSetProperty (ISMSI_HANDLE, "DOMAINNAME", svEdit1);
    MsiSetProperty (ISMSI_HANDLE, "USERNAME", svEdit2);
    MsiSetProperty (ISMSI_HANDLE, "PASSWORD", svEdit3);
    if (nResult = BACK) goto Dlg_SdWelcome;
```

6. Build your release.

Removing a Scheduled Task



Project *• This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

Task ***To remove a scheduled task from your project:***

1. In the View List under **System Configuration**, click **Scheduled Tasks**.
2. In the **Scheduled Tasks** explorer, right-click the task that you want to remove and then click **Delete**.

InstallShield removes the task from your project.

Installing, Controlling, and Configuring Windows Services

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Windows services are executable files that Windows–based systems run in the background to manage various system tasks, even if no user is currently logged in. A service is an executable file, but it must be designed as a service; you cannot automatically use an arbitrary executable file as a service. Windows services can be installed to run every time that the system starts or on demand when needed. InstallShield enables you to install new Windows services and configure existing services. Windows has a Services administrative tool with which you can view and configure the services that are installed on a system.

You can use the Services view to configure a component that installs, starts, stops, or deletes a service during installation or uninstallation. You can also use this view to configure extended service customization options that are available with Windows Installer 5. As an alternative to using the Services view, you can use the Services area under the Advanced Settings node of the Components view or the Setup Design view. If you configure service information in any of these areas, the other areas are automatically updated accordingly.

Note the following details about working with services:

- The service that you are starting, stopping, deleting, or configuring can either be already present on the target system during installation or uninstallation, or it can be installed as part of your installation.
- The service executable file should be the key file of its component. For more information, see Component Key Files.
- The Remote Installation setting for the service's component must be set to Favor Local. For more information, see Setting a Component's Remote Installation Setting.

Tip *• The View Filter at the top of the Services view lets you select a component or feature whose service data you want to display in the view, and hide the ones whose service data you do not want to display in the view. The View Filter lists your project's hierarchy of features, subfeatures, and components.*

Task ***To install, start, stop, delete, uninstall, or configure a service:***

1. If your installation is installing the service, add the service executable file to a component in your project, and make it the key file of the component. For more information, see Adding Files to Components.

   If the service is already present on target systems, skip this step.

2. In the View List under **System Configuration**, click **Services**.
3. Right-click the **Services** node and then click **Add Service**. InstallShield adds a new service.
4. Type a new name for the service now, or click it and then press F2 later to rename it.

   The name that you enter must match the name that is shown on the service's Properties dialog box. (To access an installed service's properties: In the Services administrative tool, right-click the service and then click Properties.)

5. Select the service that you added, and then configure the settings that are displayed in the right pane as needed. For information about each of the settings, see Services View.

Repeat the process for each service in this component's key file.

Note *• You must be familiar with the technical details of your service before you can configure its settings.*

Tip *• If configuring the service fails at run time, the installation may fail with an error message. ICE102 validates some of the service-related settings to help avoid such configuration failures. Therefore, it is recommended that you perform validation for your release.*

Creating Predetermined User Accounts at Run Time

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

InstallShield has support for creating one or more Windows user accounts in groups without using logon dialogs. This support requires using three Windows Installer properties for the user account, group, and password of each account that you want to be set up at run time.

Properties for Creating User Accounts

Use the following properties to create user accounts at run time:

- **ISNetApiLogonUsername**—Set the value of this property to the user account that you want the installation to create. Use either of the following formats:
  - *MachineName\UserName*
  - *DomainName\UserName*
- **ISNetApiLogonGroup**—Set the value of this property to the group to which you want the user account to belong.
- **ISNetApiLogonPassword**—Set the value of this property to the password that you want to be configured for the user account.

To configure the required properties and their values, use the Properties view. For instructions on how to do this, see Creating Properties in Windows Installer–Based Projects.

Guidelines for Creating User Accounts

To create more than one user account, separate each entry for the **ISNetApiLogonUsername** property value with a tilde that is enclosed in square brackets: [~]. For example, to create a user named **User1** on a domain named **Domain1** and a user named **User2** on a machine named **Machine2**, use the following format:

Domain1\User1[~]Machine2\User2

Each user account that you specify for the **ISNetApiLogonUsername** property must have a corresponding group in the **ISNetApiLogonGroup** property and a corresponding password in the **ISNetApiLogonPassword** property. If these three properties do not have the same number of entries, a run-time error occurs.

The following table shows sample properties and their corresponding values for the creation of three different user accounts:

Table 4-9 • Properties for User Accounts and Groups

Property Name	Property Value
ISNetApiLogonUsername	Domain1\User1[~]Domain1\User2[~]Domain2\User3
ISNetApiLogonGroup	Users[~]Users[~]Administrators
ISNetApiLogonPassword	Password1![~]Password1![~]Password1!

In this example, **User1** is part of the **Users** group and has a password of **Password1!** for the **Domain1** domain. **User2** is part of the **Users** group and has a password of **Password1!** for the **Domain1** domain. **User3** is part of the **Administrators** group and has a password of **Password1!** for the **Domain2** domain.

The **ISNetApiLogonUsername**, **ISNetApiLogonGroup**, and **ISNetApiLogonPassword** properties are formatted properties. Thus, the value of these properties can be a different property, an environment variable, a file path, or a component directory path. For example:

- **[PROPERTYNAME]**—At run time, this resolves to the value of the specified property.
- **[%EnvironmentVariable]**—At run time, this resolves to the value of the specified environment variable.
- **[#FileKey]**—At run time, this resolves to the full path of the file that has the specified file key in the **File** table.
- **[$ComponentName]**—At run time, this resolves to the installation location of the specified component.

Per-User vs. Per-Machine Installations



Project • *This information applies to Basic MSI projects.*

Two Windows Installer properties, along with the current user's privileges, affect where the configuration information such as your product's shortcuts and registry entries are stored on a target machine—to the All Users profile or the current user's profile:

- ALLUSERS determines where the configuration information is stored.
- MSIINSTALLPERUSER indicates that the Windows Installer should install the package for only the current user.

  The **MSIINSTALLPERUSER** property is available with Windows Installer 5 and on Windows 7 or Windows Server 2008 R2. Earlier versions of Windows Installer and Windows ignore this property.

You can set the **ALLUSERS** and **MSIINSTALLPERUSER** properties for your project through the Property Manager. You can also set these properties using the following methods:

- At the command line
- Through a custom action
- In the CustomerInformation and ReadyToInstall dialogs

ALLUSERS, MSIINSTALLPERUSER, and Windows 7 or Windows Server 2008 R2

If the **ALLUSERS** property is set to 2 and **MSIINSTALLPERUSER** is set to 1, the Windows Installer performs a per-user installation.

During a per-machine installation, the Windows Installer requires elevated privileges, and it directs files and registry entries to per-machine locations. If User Account Control (UAC) is available on the target system, a per-machine installation typically prompts for consent or credentials, depending on the access level of the user. During a per-user installation, the Windows Installer does not prompt for credentials, and it redirects files and registry entries to per-user locations.

For more information, see Single Package Authoring on the MSDN Web site.

Effects of ALLUSERS on Windows Vista and Later

Custom actions that have an in-script execution setting of deferred in system context are used to perform an action with the rights granted to the LocalSystem account on Windows, since the Windows Installer service runs in the system context. Actions not marked as deferred in system context run with user impersonation and have the rights that the user who launches the installation has.

When a per-user installation (that is, one where **ALLUSERS** is not set) is run, deferred-in-system-context actions run in the same context in which normal deferred or immediate custom actions run, which is with user impersonation. This can potentially cause a run-time issue with the custom action in the following circumstances:

- The user who launches the Windows Installer installation is not an administrator; or the user is running the installation on Windows Vista or later, the user is part of the Administrators group, and the user does not have administrator privileges by default.
- The custom action attempts to modify a resource in a per-machine location on the machine, such as a file in the Program Files folder, or a registry key or value in HKEY_LOCAL_MACHINE.

While this may not be an issue with Windows XP or earlier versions of Windows, Windows Vista and later do not give users full administrator privileges by default. Therefore, since a deferred-in-system-context action runs with user impersonation when **ALLUSERS** is not set, the custom action could fail.

The recommended method for preventing this behavior is to specify that a per-machine installation should always be performed by setting **ALLUSERS** to 1 in the Property Manager. Per-machine installations are generally easier to manage than per-user installations.

Default Value of ALLUSERS

The **ALLUSERS** property is set to 1 by default in all Basic MSI projects. If you configure your installation so that it can be installed per user without administrative privileges, you may want to consider changing the value of the **ALLUSERS** property or removing this property from your project.

Default Controls on the ReadyToInstall Dialog

Use the Show Per-User Option setting in the General Information view to specify whether you want to give end users the option of installing your product for all users or for only the current user. Available options for the Show Per-User Option setting are:

- **No**—InstallShield does not set any related properties. At run time, the ReadyToInstall dialog does not include the buttons that let end users specify how they want to install the product. This is the default value.
- **Yes**—InstallShield sets the **ISSupportPerUser** property equal to 1.

If the following conditions are true at run time, the ReadyToInstall dialog includes the per-user and per-machine buttons:

- The **ISSupportPerUser** property is equal to 1.
- The target system has Windows 7 or later, or Windows Server 2008 R2 or later.
- The product is not already installed on the target system.

The buttons on the ReadyToInstall dialog let end users specify how they want to install the product. If elevated privileges are required, the shield icon is included on the all-users button. If an end user selects the per-user button, the **ALLUSERS** property is set to 2, and the **MSIINSTALLPERUSER** property is set to 1. If an end user selects the all-users button, the **ALLUSERS** property is set to 1, and the **MSIINSTALLPERUSER** property is not set.



***Note** • Selecting No for the Show Per-User Option setting in the General Information view does not prevent end users from setting **MSIINSTALLPERUSER** from the command line when they run your installation. If your installation does not support this, you may want to add a launch condition or other run-time check to prevent this from occurring.*

Default Controls on the CustomerInformation Dialog

By default, the CustomerInformation dialog in all Basic MSI projects does not display the radio button group that enables end users to specify whether they want to install the product for all users or for only the current user. This is the recommended implementation for this dialog.

Task ***To display the radio button group that lets end users set ALLUSERS from the CustomerInformation dialog:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, under **All Dialogs**, expand the **CustomerInformation** dialog, and then click **Behavior**. InstallShield displays the list of controls for the CustomerInformation dialog.
3. At the bottom of the lower-right pane, click the **Conditions** tab. In the upper-right pane, InstallShield displays the conditions for the selected control.
4. In the list of controls, click **DlgRadioGroupText**. InstallShield displays this control's conditions in the upper-right pane.
5. In the upper-right pane, right-click the row that contains the condition whose value is 1 and whose action is Hide, and then click **Delete**.
6. Repeat steps 4 and 5 for the **RadioGroup** control.

Running InstallScript Installations Without Administrative Privileges

Project • *This information applies to InstallScript projects.*

InstallShield enables you to create InstallScript installations that can be run by end users who do not have administrative privileges. The proper locations of registry entries, folders, and Start Menu items are dynamically determined at run time. The InstallScript system variable ALLUSERS is the key to such installations; if an installation is run by an end user who does not have administrative privileges, ALLUSERS is initialized to FALSE, and assigning it a new value in the script has no effect.

Note the following details:

- Automatic registration of files (which are included in components that have Yes selected for the Self-Register setting) requires administrative privileges.
- InstallScript objects typically require administrative privileges, since they usually install or register files in locations that require administrative privileges.
- Installing a DIFx driver requires administrative privileges.
- Installing the Setup Player for a One-Click Install installation requires administrative privileges.
- Installing a Windows service requires administrative privileges.
- Connecting to a remote registry requires administrative privileges.
- If an end user who is not an administrator or power user attempts to maintain an InstallScript installation that was installed when ALLUSERS was set to TRUE, the installation displays an error message during initialization. The error message indicates that a user with administrator rights installed the product, and similar privileges are needed to modify or uninstall the product.

Customizing Installation Behavior

An important aspect of creating an installation is customizing it for your end users' needs. The "Customizing Installation Behavior" section of the documentation discusses various features of InstallShield that help you extend the functionality of your installation. For example, you may find it useful to create custom actions to add support for something not directly supported by Windows Installer. Furthermore, you may set up custom actions to call any script that you write in the InstallScript view's script editor. Refer to this section of the documentation for more information on how you can customize the installation behavior in your project.

Using InstallScript

You can leverage the power and ease of InstallScript to extend the functionality of your installation package. The InstallScript view includes a script editor pane for you to author your InstallScript code.



Project • *You must have a file named* **Setup.rul** *in your project if you are using InstallScript. InstallScript and InstallScript MSI projects contain a* **Setup.rul** *file by default. You must add a* **Setup.rul** *file in Basic MSI, DIM, and Merge Module projects if you want to use InstallScript custom actions in these types of projects.*

Using Event-Driven InstallScript in InstallScript and InstallScript MSI Projects

InstallScript and InstallScript MSI projects contain an event-driven script. When you use InstallScript in these project types, many of the functions are logged for uninstallation.

Using InstallScript in Custom Actions

Basic MSI, DIM, and Merge Module Projects

In a Basic MSI, DIM, or Merge Module project, you can use custom actions to run InstallScript at run time. These project types do not support event-driven script.

InstallScript MSI Projects

In an InstallScript MSI project, you can use an InstallScript custom action to extend functionality in an execute sequence where the default event handlers are not scheduled appropriately for your installation's needs.

Creating and Using Custom Actions



Task ***To author an InstallScript custom action and execute it in your installation:***

1. Add a blank Setup.rul to your project in the InstallScript view if it does not already exist.
2. Write an entry-point function. Note that you cannot call an entry-point function in the User Interface sequence for an InstallScript MSI installation project.
3. Compile the script.
4. Create a custom action that calls your InstallScript function.

5. Invoke the InstallScript custom action by either including it in a sequence (InstallScript MSI, Basic MSI, and merge module projects) or executing it as a control event (Basic MSI and merge module projects).
6. Debug if necessary.

Note *• As with other custom actions, changes made to the system through InstallScript custom actions are not automatically restored when the package is uninstalled. Because InstallScript custom actions are not logged and removed by the uninstaller, you must write a corresponding custom action to uninstall any changes that your custom action makes.*

Overview of ISSetup.dll

ISSetup.dll is a C++ MSI DLL that contains the full InstallScript scripting run-time engine. For InstallScript MSI, Basic MSI, DIM, and Merge Module projects, **ISSetup.dll** executes InstallScript custom actions. For InstallScript projects, **ISSetup.dll** must be in the Disk1 folder.

Your installation always uses the InstallScript scripting run-time engine with which it was built, even if more a more recent version is installed on a target machine by another installation.

Important *• Note the following information:*

- **ISSetup.dll** *allows you to add only 1,000 InstallScript custom actions to your InstallScript MSI, Basic MSI, DIM, or Merge Module project. If your project includes more than 1,000 InstallScript custom actions, a build error is generated.*
- *The version of* **ISSetup.dll** *that is used for InstallScript projects is different than the one that is used in InstallScript MSI, Basic MSI, DIM, and Merge Module projects. These two versions are not interchangeable.*

Script Files

Project *• This information applies to InstallScript projects.*

When you create an installation project, InstallShield creates two script files and stores them in the **Script Files** folder of your project folder.

- **Setup.rul** is created for global event handlers and exception handlers.
- **FeatureEvents.rul** is created for feature event handlers.

Initially, these two files are empty; the default event handlers defined by InstallShield do not appear in these files unless you select them from within the InstallScript view in InstallShield. When you do so, they are inserted into the appropriate script file and displayed in the script pane in the InstallScript view, where you can edit them.

Note that if you change the default feature event handler code in **FeatureEvents.rul**, you must put the following statement in **Setup.rul** to include your changes in the installation:

```
#include "FeatureEvents.rul"
```

Creating InstallScript Files



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

You must have a file named **Setup.rul** *in your project if you are using InstallScript. InstallScript and InstallScript MSI projects contain a* **Setup.rul** *file by default. You must add a* **Setup.rul** *file in Basic MSI, DIM, and Merge Module projects if you want to use InstallScript custom actions in these types of projects.*



Task ***To add a new InstallScript file to your project:***

1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. In the **InstallScript** explorer, right-click **Files** and click **New Script File**.
3. Name the file.

New script files are named **Setup.rul** by default. If you already have a file called **Setup.rul**, a new file is added with the name Setup*n*.rul, where *n* is a successive number. You can rename the file by right-clicking it and then clicking Rename.

The new script file is placed in the Link To folder. InstallShield attempts to use a path variable in case you move your project. You cannot edit the Link To value.

You can also include additional header files (.h files) and script files. Repeat the above procedure to add a new script file to your project.

Opening an InstallScript File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Task ***To open a script file in your project so that you can edit it:***

1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. In the **InstallScript** explorer, click the file that you want to open.

Inserting and Importing Script Files

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

InstallShield enables you to reuse InstallScript files (.rul) and InstallScript header files (.h) in multiple projects. You can insert or import script files into a project through the InstallScript view:

- Inserting a script file creates a link to the script file in its current location.
- Importing a script file copies the script file to the folder containing the script files for your project. The script files that you import can be stored somewhere on your system, or they can be stored in a repository.

InstallShield supports all path variable types that you define in the Path Variables view for the location of these script files. However, it is important to be aware that a corresponding folder structure exists in your source code database so that your source code control software can resolve paths.

Inserting Script Files

Task ***To insert a script file:***

1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. In the **InstallScript** explorer, right-click **Files** and then click **Insert Script Files**. The **Open** dialog box opens.
3. Select the **InstallScript** file (.rul) or InstallScript header file (.h) that you want to insert.
4. Click **Open**.

Importing Script Files

Task ***To import a script file:***

1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. In the **InstallScript** explorer, right-click **Files** and then click **Import Script Files**. The Import InstallScript Files dialog box opens.
3. Do one of the following:
   - In the **Repository Items** box, click the InstallScript file (.rul) or InstallScript header file (.h) that you want to add to your project.
   - If the script file that you want to import is not stored in the repository, click the **Browse** button to select it.
4. Click **OK**.

Compiling Scripts

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

You must compile your script before your InstallScript code can be called in your installation.

InstallShield searches only for a file named **Setup.rul** when compiling the script. You can include files with different names, but they must be included in **Setup.rul** or in an include file with the #include preprocessor statement.

Task ***To compile your script, do one of the following:***

- On the **Build** menu, click **Compile**.
- Click the **Compile** button on the toolbar.
- Press CTRL+F7.

Before compiling, InstallShield saves any changes that you have made to your script files. The compiler's status, including any error or warning messages, is displayed in the Output window. Double-click a compiler message to go to the line in your script where the error was found.

If you compile your script file after making changes to it, it is not necessary to rebuild your release. Note that InstallShield automatically compiles your script whenever you build a release.

If your script compiled successfully, InstallShield creates **Setup.inx** (the object code that the setup engine executes) and streams it into your Windows Installer package when you build a release.

You may also need to uninstall the release that you previously ran to test your InstallScript custom action before you can see the changes to the script.

You can set compiler options on the Compile/Link tab of the Settings dialog box.

Debugging Scripts

The InstallScript Debugger is useful for stepping through your InstallScript code and checking the progress of your code.

Before you can debug your script, you must first compile it, execute it as a custom action (if applicable), and build a release.



Task ***To debug your script directly from within InstallShield, do one of the following:***

- On the **Build** menu, click **Debug**.
- Press F5.
- Click the **Debug** button on the toolbar.

InstallShield will run the installation and open the InstallScript Debugger when the custom action is executed.

While you are in the InstallScript Debugger, press F1 at any point to view the InstallScript Debugger Help.

For information about debugging on a test system, see Debugging an Installation on any Computer.

Using Preprocessor Statements to Debug the Script

Use the #define and #ifdef statements to create an internal debugger in the script.



Task ***To deug a script by using the preprocessor statements:***

1. Wherever you want to insert a debug statement in the script, start with the following #ifdef directive:

   ```
   #ifdef DEBUG
   ```

2. On lines that follow that directive, type the debug statements.
3. On a separate line after the debug statements, type:

   ```
   #endif
   ```

4. For debugging purposes, compile the following compiler setting:

   ```
   DDEBUG=1
   ```

Here is an example of a debugging section using an #ifdef statement:

```
#ifdef DEBUG
    if nResult < 0 then
        WriteLine (LogFileHandle, "PlaceBitmap failed");
    endif;
#endif
```

The InstallScript Debugger enables you to trace program execution and inspect variables as your installation executes.

Renaming an InstallScript File



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*



Task ***To rename an InstallScript file:***

1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. In the **InstallScript** explorer, right-click the file and click **Rename**.
3. Type a new name for the file.



Project** • InstallScript projects must contain a source file named* **Setup.rul; that file is the main compilation unit of an installation script. If your InstallScript project does not include a script file with that name, error C8503 occurs when you compile your script or build a release.*

Using String Entries in Scripts

You can use string identifiers in your script in place of any value that accepts a string literal. When the custom action is executed, the installation replaces the string identifier with the corresponding string value for the language in which the installation is running.

String identifiers must follow an at symbol (@) in your script. The Select String dialog box lets you browse the list of string entries in your project. It also lets you modify string entries for the default language before inserting the selected string identifier into your script.

For example, assuming your project contains a string identifier called MSG_ACTION_SUCCEEDED, you could display its value in a message box as demonstrated below:

```
szMsg = @MSG_ACTION_SUCCEEDED;
nType = INFORMATION;
MessageBox (szMsg, nType);
```

Removing InstallScript Files from Projects



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

When you remove a script file from your installation, you are deleting the reference to that file from your project, but not the file itself. If you later decide to insert it into your installation, you do not need to rewrite your script.

Note that the script file may, in fact, still be compiled if it is included in another file.



Task ***To remove InstallScript files from your project:***

1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. In the **InstallScript** explorer, right-click the file and click **Remove**.



Caution *• You cannot rename a file to the name of an existing file under the project location, even if the script file has been removed through the above procedure. For example, assume you have a script file called* **Setup.rul** *in your project. Next, remove that file and add a new script file, called* **Script1.rul** *by default. You cannot rename* **Script1.rul** *to* **Setup.rul***, because doing so would overwrite the first* **Setup.rul***. To avoid this, you should move your original* **Setup.rul** *out of your project folder, rename it, or delete it through Windows Explorer.*

Creating Script Libraries (.obl Files)

InstallScript's built-in functions are defined in library files (.obl files) to which a script is linked when the script is compiled.



Task ***To create a library file for functions that you have defined:***

1. Create one or more .rul files containing the definitions of your functions.
2. At the command line, for each of your .rul files, run **Compile.exe** with the `-c` switch to compile the file without linking it to any existing library file. This will create an .obs file (rather than the .inx file that is created by compiling without the `-c` switch). For example, enter the following command line to create the file **MyFunc.obs** in the current folder.

   ```
   Compile MyFunc.rul -c
   ```

   To create an .obs file with a different name or location, use the `-o` switch.
3. Run **Compile.exe** with the `-l` switch, and one or more .obs files as parameters, to create the library file. For example, enter the following command line to create the file **MyFunc.obl** in the current folder.

   ```
   Compile MyFunc.obs -l
   ```

   Entering the following command line creates the file **MyFunc1.obl** in the current folder.

   ```
   Compile MyFunc1.obs MyFunc2.obs -l
   ```

To create an .obs file with a different name or location, use the `-o` switch.

Tip *• If you have many .obs files, you can shorten the command line by using a command file as in the following example:*

```
Compile @MyObsFiles.txt -l
```

To quickly create the command file, you can use the MS-DOS command DIR with its `/b` *(bare format) switch and redirect the output to a file. For example:*

```
DIR *.obs /b > MyObsFiles.txt
```

To compile an installation script using your library file, specify the library file on the command line or—when compiling within InstallShield—on the Compile/Link Tab of the Settings dialog box.

Publishing InstallScript Files (.rul and .h) to a Repository

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

If you have an existing InstallScript file (.rul) or InstallScript header file (.h) that you would like to reuse in other projects or share with other users, you can publish it to a repository.

Task ***To publish a script file to a repository:***

1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. In the **InstallScript** explorer, right-click the script file that you would like to publish, and then click **Publish Wizard**. The **Publish Wizard** opens.
3. Complete the panels in the Publish Wizard.

After you have imported a script file from a repository into a project, there is no link between the current script file and the existing repository script file. If you make a change to the script file and then republish it to the repository, it does not affect the script file in the project to which it was imported. However, you can reimport the script file from the repository into your project.

InstallScript Lists

Lists are used to store related information, such as strings or numbers. InstallScript lists are very similar to single-linked lists in the C language. InstallScript list functions are very flexible, enabling you to return information in an order different from the order in which it was stored and access and use that information in a variety of ways.

List Functions

InstallShield provides a number of functions for creating and manipulating lists. There are three types of InstallScript list functions:

- Functions ending in **-String** that work with string lists only
- Functions ending in **-Item** that work with number lists only
- Functions that work with either string or number lists

InstallShield also has many secondary list-related functions that use or create lists.

List Structure

Lists are used to store related information—either strings or numbers. All the information in a list must be of the same data type, and the number of elements is limited only by the available memory.

An InstallScript list has two parts. The first part is the head, which InstallShield uses internally. The head of the list contains general information about the list, such as whether it contains strings or numbers. The head also contains pointers to the beginning and end of the list.

The second part of the list is the list body. The list body contains the actual strings or numbers. You can have as many strings or numbers in a list as the memory in the system will allow.

Remember that lists cannot contain both numbers and strings. A list must have only strings or only numbers.

Variables representing lists can be declared as type LIST or type LONG. Lists exist only in memory, meaning they are destroyed when the installation is complete. If a list is local to a function, the list is destroyed when the function returns control to the calling code.

Creating and Destroying Lists

Before creating a list, decide which type of list you want to build: a string list or a number (item) list. To create the list, call the **ListCreate** function:

```
// This builds the list head for a string list.
listID1 = ListCreate (STRINGLIST);
```

–or–

```
// This builds the list head for a number (item) list.
listID2 = ListCreate (NUMBERLIST);
```

ListCreate automatically builds the head of the list and returns its ID number. The ID is used in all subsequent functions that operate on the list. Therefore, you must always create a list using **ListCreate** before you use any other list function. You must store the return value from **ListCreate** in a variable of type LIST or type LONG.

This fragment creates a number list and then a string list. It also tests each one to make sure that the lists were created successfully.

```
// This creates an empty list for strings.
listID1 = ListCreate (STRINGLIST);

if (listID1 = LIST_NULL) then
      MessageBox ("Unable to create the string list", SEVERE);
endif;
```

```
// This will create an empty list for numbers.
listID2 = ListCreate (NUMBERLIST);

if (listID2 = LIST_NULL) then
    MessageBox ("Unable to create the number list", SEVERE);
endif;
```

When you are finished using a list, you will typically want to destroy the list to free the memory for other uses. **ListDestroy** destroys the list and its contents. This example creates a list referenced by listID, adds a string to the list, and then destroys the entire list.

```
listID = ListCreate (STRINGLIST);

if (listID = LIST_NULL) then
    MessageBox ("Unable to create list.", SEVERE);
    abort;
endif;

ListAddString (listID, "This is a string in the list", AFTER);
ListDestroy (listID);
```

If you do not destroy a list using **ListDestroy**, the list will be destroyed when the installation is complete. If the list is local to a function, the list is destroyed when the function returns control to the calling code.

Adding Elements to Lists

InstallShield provides several functions for adding elements to lists:

Table 4-1 • Functions that Are Used to Add Elements to Lists

Function	Description
ListAddItem	Adds an item to the list.
ListAddString	Adds a string to the list.
ListReadFromFile	Reads a text file into a list.

ListAddString and **ListAddItem** add a single element to the list that you specify. Remember that, regardless of where you place the new string in the list, it becomes the current string. Use the options BEFORE and AFTER to indicate where you want to place the new element in the list relative to the current element. If you are working with a newly created list, using either BEFORE or AFTER will add the string to the first element position in the list.

Adding elements to a list and the resulting effects on the list order and the element in the current position are most easily explained by example. The examples below use string lists and **ListAddString**, but the same principles and steps apply to using **ListAddItem** and number lists. Consider these scenarios:

- Adding an element to an empty list
- Adding an element before the current element
- Adding an element after the current element
- Adding elements before and after the current element

Adding an Element to an Empty List

The first string that you add to the list goes immediately after the head of the list. This string (*String 1* in the sample code) becomes the current string in the list. The script fragment shown below results in a list like that depicted after it:

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
else
    // Add a string to the list.
    szString = "String 1";
    ListAddString (listID, szString, AFTER);
endif;
```

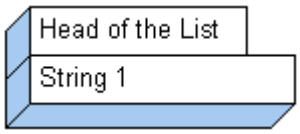


Figure 4-1: String 1 is added to a list.

Adding an Element Before the Current Element

If the current string is the first string in the list and you add a new string before it, the new string becomes the first string in the list. The string that was formerly in the first element position now resides in the second element position. The new string at the first element position is now the current string:

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
else
    // Add some strings to the list.
    szString = "String 1";
    ListAddString (listID, szString, AFTER);

    szString = "String 2";
    ListAddString (listID, szString, BEFORE);
endif;
```

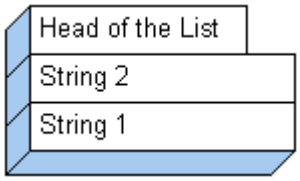


Figure 4-2: String 2 is added before String 1.

Adding an Element After the Current Element

If the current string is the first string in the list, and you add the new string after the current string, the new string becomes the second string in the list, as well as the new current string. Refer to the script fragment below and to the illustration following it:

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
else
    // Add some strings to the list.
    szString = "String 1";
    ListAddString (listID, szString, AFTER);

    szString = "String 2";
    ListAddString (listID, szString, AFTER);
endif;
```

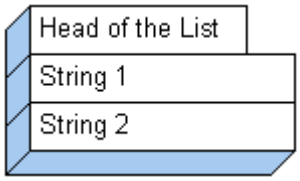


Figure 4-3: String 2 is added after String 1.

Adding Elements Before and After the Current Element

As another example, the code segment shown below creates a new list and puts String 1 in the first position. String 2 is then added before String 1, leaving String 2 in the first position as the current string. Next, String 3 is added after the current string, resulting in the list depicted below.

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
else
    // Add some strings to the list.
    szString = "String 1";
    ListAddString (listID, szString, AFTER);

    szString = "String 2";
    ListAddString (listID, szString, BEFORE);

    szString = "String 3";
    ListAddString (listID, szString, AFTER);
endif;
```

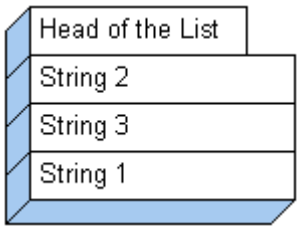


Figure 4-4: String 1 is added. Then String 2 is added before String 1. Last, String 3 is added after String 2.

In the example above, if String 3 were added before the current string, the result would be as shown below, with String 3 becoming the current string. This code segment is shown below:

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
else
    // Add some strings to the list.
    szString = "String 1";
    ListAddString (listID, szString, AFTER);

    szString = "String 2";
    ListAddString (listID, szString, BEFORE);

    szString = "String 3";
    ListAddString (listID, szString, BEFORE);
endif;
```

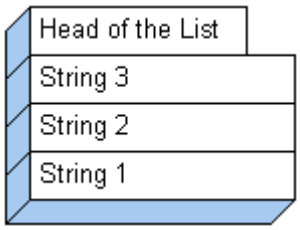


Figure 4-5: String 1 is added. Then String 2 is added before String 1. Last, String 3 is added before String 2.

Changing Existing Elements in a List

Call the **ListSetCurrentString** function to change the value of an element in a string list. Remember that only the current element may be changed, so be sure to make the string you want to update the current string in the list.

Call the **ListSetCurrentItem** function to change the value of an element in a number list. Again, the item that you want to update must be the current element in the list.

The example below demonstrates calling **ListSetCurrentItem** to change the value of the current item in a number list. The **ListSetCurrentString** function works in the same manner, but with a string list and string variables.

```
// Create a list and verify its creation.
listID = ListCreate (NUMBERLIST);
if (listID = LIST_NULL) then
    MessageBox ("Unable to create list.", SEVERE);
```

```
    abort;
endif;


// Add items (1078 and 304) to the list.
nItem = 1078;
ListAddItem (listID, nItem, AFTER);
nItem = 304;
ListAddItem (listID, nItem, AFTER);

// Current item is the second item (304).
// Now set current item to new value (305).
nItem = 305;
ListSetCurrentItem (listID, nItem);
ListDestroy (listID);
```

Deleting Elements from a List

Call the **ListDeleteString** function to delete the current string from a list. Or, call the **ListDeleteItem** function to delete the current number from a list. If there are no more elements to delete, these functions return the END_OF_LIST constant.

Note that since **ListDeleteString** and **ListDeleteItem** delete the current element, you must reset the current element to the element that you want deleted. After deletion, the next element in the list becomes the current element. You reset the element by any of the methods described in Traversing Lists.

The example below illustrates the use of several list functions, including **ListDeleteString**. The **ListDeleteItem** function is used in the same manner, except that the list is a number list and the variables are number variables.

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

if (listID = LIST_NULL) then // Test for a valid list.
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
szString = "String 1";
ListAddString (listID, szString, AFTER);
szString = "String 2";
ListAddString (listID, szString, AFTER);
szString = "String 3";
ListAddString (listID, szString, AFTER);

// Delete the current string.
ListDeleteString (listID);

// Reset the current string in the list.
lResult = ListCurrentString (listID, svString);

// svString contains "String 2."
```

Finding a Particular Element in a List

You can traverse lists non-incrementally. Call the **ListFindString** function or the **ListFindItem** function when you want to search for a specific string or number element in a list. These two functions begin their search at the current element and continue forward through the list from that point.

To start a search from the beginning of a list, call the **ListGetFirstString** or the **ListGetFirstItem** function before calling the **ListFindString** or the **ListFindItem** function.

When the **ListFindString** or the **ListFindItem** function finds the specified string or number, it becomes the current element in the list.

In the script fragment below, a number list is created, and the number 1 (in nItem) is added as the first element. Then, **ListFindItem** searches the list for the number 1, and deletes it, if found. Finally, the list is destroyed.

```
listID = ListCreate (NUMBERLIST);

if (listID = LIST_NULL) then
    MessageBox ("Unable to create list.", SEVERE);
    abort;
endif;

nItem = 1;
ListAddItem (listID, nItem, AFTER);

if (ListFindItem (listID, nItem) = 0) then
     ListDeleteItem (listID);
endif;

ListDestroy (listID);
```



Note *• The* ***ListFindString*** *and* ***ListFindItem*** *functions look for only the first instance of the specified string or number at or after the current element.*

Getting the First and Next Elements in a List

Call the **ListGetFirstString** function or the **ListGetFirstItem** function to return the first string element or number element, respectively, from a list. The element that you retrieve then becomes the current element in the list.

Call the **ListGetNextString** function or the **ListGetNextItem** function to return the string element or number element after the current element in a list. The element that you retrieve then becomes the current element in the list.

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list.
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
ListAddString (listID, "String 1", AFTER);
ListAddString (listID, "String 2", AFTER);
ListAddString (listID, "String 3", AFTER);
```

```
// Traverse the list and display the strings in a message box.
lResult = ListGetFirstString(listID, szDriveName);

while (lResult != END_OF_LIST)
    MessageBox (szDriveName, INFORMATION);
    lResult = ListGetNextString (listID, szDriveName);
endwhile;
```

Reading a File into a List

Call the **ListReadFromFile** function to read an entire file into a string list. Each line in the file becomes an element in the list. The **ListReadFromFile** function provides an easy way to load a list, rather than building it one item at a time.

Setting an Index in a List

InstallShield provides the **ListSetIndex** function, which lets you make an element the current element using an index number. If you know the location of a particular element in a list, you can call the **ListSetIndex** function to access that element immediately. You can traverse a list in either direction by using the index to set a specific element in a list to the current element. The index of the list starts at 0 (zero).

The **ListSetIndex** function works on both string and number lists. After you set the indexed element as the current element, call either the **ListCurrentItem** function or the **ListCurrentString** function to return the value of the indexed item.

This example demonstrates traversing a list non-incrementally using **ListSetIndex**.

```
listID = ListCreate (STRINGLIST);
GetGroupNameList (listID);
nCheck = ListSetIndex (listID, LISTFIRST);

while (nCheck != END_OF_LIST)
    ListCurrentString (listID, svString);
     MessageBox (svString, INFORMATION);
     nCheck = ListSetIndex (listID, LISTNEXT);
 endwhile;

ListDestroy (listID);
```

The **ListCount** function tells you how many elements are in a list. The **ListCount** function is used mainly for general information purposes, although it can be used to establish an upper index value in conjunction with **ListSetIndex**. For example, you can call **ListCount** to get the number of elements in a list, and use that value with **ListSetIndex** to traverse a list. The above example, which uses a while loop, is rewritten below using an InstallScript for loop based on the number of elements in the list.

```
listID = ListCreate (STRINGLIST);
GetGroupNameList (listID);

// Get the number of elements in the list.
nItems = ListCount (listID);

// Display the number of elements in the list.
SprintfBox (INFORMATION, "", "i = %d", nItems);

// Loop for nItems times beginning with zero,
// displaying each list element in turn in a message box.
```

```
for i = 0 to (nItems - 1)
    ListSetIndex (listID, i);
    ListCurrentString (listID, svString);
    MessageBox (svString, INFORMATION);
    nCheck = ListSetIndex (listID, LISTNEXT);
endfor;

ListDestroy (listID);
```

Traversing Lists

InstallShield provides these functions for traversing lists incrementally and non-incrementally:

Table 4-2 • Functions that Are Used to Traverse Lists

Function	Description
ListCount	Sees how many string or numeric elements are contained in a specified list.
ListCurrentItem	Returns the current item in the list.
ListCurrentString	Returns the current string in the list.
ListFindItem	Attempts to find a numeric element in a list. If found, the element becomes the current element of the list.
ListFindString	Attempts to find a string element in a list. If found, the element becomes the current element of the list.
ListGetFirstItem	Retrieves the first element from a number list.
ListGetFirstString	Retrieves the first string from a string list.
ListGetNextItem	Retrieves the element after the current element from a number list.
ListGetNextString	Retrieves the element after the current element from a string list.
ListSetIndex	Sets the current element of the list as an index.

InstallShield uses single-linked lists, which means that unless you use functions that set indices or search for specific elements in lists, you can traverse lists incrementally in one direction only: from the first element to the last.

InstallShield allows you to traverse lists in non-incremental fashion by means of indices and by searching for particular elements in lists. Refer to the individual function descriptions for more details.

Most list traversing and list access operations are carried out relative to the current list element. Furthermore, most of the functions used to traverse and access lists establish a current element as a result of their action. Therefore, making an element the current element in a list is not an isolated action; it is a byproduct of another action.

If a list is empty, adding an element to the list will establish a current element. If a list is not empty, then making an element the current element is best accomplished by traversing the list or searching for a particular element in the list.



***Tip** • Lists are often processed within while loops, usually checking for END_OF_LIST. An infinite loop can result if the list is not valid. If you are processing lists in a while loop, make sure that you have created the list with the **ListCreate** function, and that you have not destroyed the list with the **ListDestroy** function.*

Writing a List to a File

Call the **ListWriteToFile** function to write the contents of a string list to a file. Each element in the list becomes a line in the file.

The example script below reads the Autoexec.bat file into the listFile string list and then writes that string list to a file named Autoexec.bak.

```
listFile = ListCreate (STRINGLIST);
szPath = "C:\\";
szFileOld = "C:\\Autoexec.bat";
szFileNew = "C:\\Autoexec.bak";

if (ListReadFromFile (listFile, szFileOld) < 0) then
   MessageBox ("ListReadFromFile failed.", SEVERE);
endif;

ListWriteToFile (listFile, szFileNew);
```

Saving InstallScript Files

All open script files are saved automatically when you click Save on the File menu in InstallShield.

System Restore



***Project** • This information applies to InstallScript projects.*

System Restore is a Windows feature that enables end users to restore PCs corrupted during software installation. The System Restore feature automatically monitors and records key system changes to the end user's PC. System Restore reduces support costs and increases customer satisfaction by letting the end user easily undo a change that may have harmed their system or revert to a time when they knew that their system was performing optimally.

InstallScript installations support System Restore by setting a *restore point* before starting the file transfer; the end user can then use System Restore to restore the system to the state it was in before the file transfer.



***Note** • Installation actions (for example, registry changes and file modifications) that take place before file transfer cannot be undone by System Restore.*

Your installations are System Restore compatible by default. You can disable System Restore compatibility by placing the following code in your script's OnBegin event handler:

```
Disable( PCRESTORE );
```

If the file **Wininit.ini** exists in the target machine's **Windows** folder, the installation cannot set a restore point. To handle **Wininit.ini**, put code like the following in the **OnFirstUIBefore** and **OnMaintUIBefore** event handler functions:

```
/* Look for Wininit.ini in the Windows folder. If it is found ... */
if FindFile( WINDIR, "Wininit.ini", svResult )=0 then
    bRebootForSystemRestore = TRUE;
    /* ... get its size. */
    GetFileInfo( WINDIR ^ "Wininit.ini", FILE_SIZE, nvSize, svResult );
    /* If its size is zero bytes ... */
    if nvSize=0 then
        /* ... delete Wininit.ini. */
        if DeleteFile( WINDIR ^ "Wininit.ini" )=0 then
            bRebootForSystemRestore = FALSE;
        endif;
    endif;
    /* If Wininit.ini has a non-zero size or could not be deleted, notify the end user and allow a
reboot. */
    if bRebootForSystemRestore then
        szQuestion = "Windows System Restore lets you undo " +
            "changes to your computer. If you want to be able " +
            "to use System Restore to undo this installation, " +
            "you must reboot your computer now.\n\n" +
            "Do you want to reboot your computer now?"
        if AskYesNo( szQuestion, YES )=YES then
            System( SYS_BOOTMACHINE );
        endif;
    endif;
endif;
```

Getting and Setting Properties



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Windows Installer properties can contain useful information about the product, the setup, the operating system, and the user. By calling **MsiGetProperty** and **MsiSetProperty**, you can directly interact with these properties in your immediate InstallScript custom action.



Note *• Note that the **MsiGetProperty** and **MsiSetProperty** properties cannot be used for deferred InstallScript custom actions, which do not have access to the active .msi database and do not recognize any Windows Installer properties. They can access only the information that has been written into the execution script.*

The following example retrieves the user name from the Windows Installer setup package, confirms the value, gives the user the opportunity to change it, and then writes the new value back into the database.



Note • *It may be necessary to specify a proper buffer size if the property value exceeds 1024 characters. For an example of how to write code for this scenario, refer to Changes in Behavior for Some MSI APIs That Are Called in InstallScript Custom Actions in the "Upgrading Projects from InstallShield 2011 or Earlier" topic.*

```
// Include header file for built-in functions
#include "isrt.h"
// Include header file for MSI API functions and constants
#include "iswi.h"

export prototype Func1(HWND);

function Func1(hMSI)
    STRING svName;
    NUMBER nvSize, nResponse;
begin
    // Retrieve the user's name from the MSI database
    nvSize = 256;
    MsiGetProperty (hMSI, "USERNAME", svName, nvSize);

    nResponse = AskYesNo ("Your name will be registered as " +
                          svName + ". Is this correct?", YES);

    if nResponse = NO then
        AskText ("Enter the name that will be registered for " +
                 "this product.", svName, svName);
        MsiSetProperty(hMSI, "USERNAME", svName);
    endif;
end;
```

Using Bit Flags



Project • *This information applies to InstallScript projects.*

Bit flags are one or more (up to 32) Boolean values stored in a single number variable.

Each bit flag typically has a corresponding predefined constant associated with it; this constant has the bit for this flag set to 1 and all other bits set to 0. For example, the following constant identifies the bit flag for bit 0, that is, the right-most bit in the number:

```
#define BITFLAG_EXAMPLE_1 0x00000001
```

Other bit flags could be the following:

```
#define BITFLAG_EXAMPLE_2 0x00000002
#define BITFLAG_EXAMPLE_3 0x00000004
```

Note that 0x00000003 is not a valid bit flag, since this value corresponds to two bits in the number being set to 1.

Setting Bit Flags in a Variable

To set bit flags in a variable, use the bitwise OR operator (|). For example, to assign the value BITFLAG_EXAMPLE_1 and BITFLAG_EXAMPLE_2 to the number variable *nFLags*, use code like the following:

```
nFlags = nFlags | BITFLAG_TEST_1 | BITFLAG_TEST_2;
```

Clearing a Bit Flag from a Variable

To clear a bit flag from a variable, combine the bitwise AND operator (&) with the bitwise NOT operator (~). For example, to remove the BITFLAG_TEST_1 flag from nFlags, use code such as the following:

```
nFlags = nFlags & ~BITFLAG_TEST_1;
```

Testing a Variable for a Bit Flag

To test a variable for a bit flag, use the bitwise AND operator (&) and the bit flag constant to zero all the bits other than the bit for which you are testing. For example, if *nFlags* is currently set to **BITFLAG_TEST_1 | BITFLAG_TEST_2**, the following expression evaluates to true (that is, a non-zero result):

```
nFlags & BITFLAG_TEST_1
```

All bits other than the rightmost bit are set to 0 by the & operation; the rightmost bit is 1 since it is 1 in both the constant BITFLAG_TEST_1 and *nFlags*.

The following expression evaluates to FALSE since the third bit of nFlags is 0 and all other bits are set to 0 by the & operation:

```
nFlags & BITFLAG_TEST_3
```

String Comparisons

When InstallShield compares two strings, it starts by comparing the initial character in the first string with the initial character in the second string. If those characters are equal, InstallShield then compares the characters in the next position of each string. If those characters are also equal, it moves on to the characters in the next position, continuing in sequence until it encounters one of the following conditions:

- Two characters in the same relative position in the two strings do not match. In this case, InstallShield bases its resolution on the comparison of those two characters. If the character in string one has a greater value than the character in the corresponding position of string two, then string one is greater; otherwise string two is greater.
- The end of one string is encountered without finding unequal characters in corresponding positions. In this case, the strings are of unequal length and therefore they are not equal.
- The end of both strings is reached without finding unequal characters in corresponding positions. In this case, the strings are equal in length and all characters match; therefore the strings are equal.

Consider the following example:

```
svString1 = "trusting";
    svString2 = "TRUTHFUL";

    if svString1 = svString2 then
        MessageBox("Equal", INFORMATION);
    else
        MessageBox("Not Equal", INFORMATION);
    endif;
```

String comparisons are not case-sensitive. Because an uppercase character is equal to its lowercase counterpart, InstallShield finds that the first three characters of **trusting** and the first three characters of **TRUTHFUL** are equivalent. The comparison ends with the test of the characters in the fourth position of each string. Since **s** is lower than **t** in the ASCII character table, svString1 does not equal svString2. The remaining characters are not compared and the else branch is executed.



***Note** • The value of each character is based on its ASCII value. For information about the ASCII values of specific characters and symbols, refer to a basic programming manual.*

Using Null-Delimited Strings



***Project** • This information applies to InstallScript projects.*

Note the following when using a null-delimited, double-null-terminated string (for example, **abc\0def\0ghi\0\0**; such strings are also sometimes referred to as *multiple null-delimited strings* or *multiple null-terminated strings*):

- When declaring a variable that will be set to a null-delimited string value, explicitly size the variable; for example:

  ```
  STRING szString[1024];
  ```

  Do not use autosizing; the installation engine expects autosized strings to not include null characters within the string.

- Do not try to create arrays of null-delimited strings; since array elements must always be autosized, string arrays do not currently support this type of string.
- The specified size of a null-delimited string should be at least the number of characters to be stored plus two for the two terminating null characters.
- Since strings are automatically initialized to all null characters, you do not need to explicitly set the second-to-last character to null (though this will not cause any adverse effects) unless you have previously set the second-to-last character to something other than null.
- The best way to determine the length of a null-delimited string is to call the **CharReplace** function to replace the null characters and then call the **StrLengthChars** function to determine the size of the resulting string.
- The **StrGetTokens** and **StrPutTokens** functions may be useful when working with null-delimited strings.
- Most InstallScript functions other than the ones noted above do not work with multiple null-delimited strings. If you want to use a multiple null-delimited string with a built-in function, use the **CharReplace** function first to replace the null characters with another character, such as an underscore (_).

Relative Path

A relative path includes all of the information necessary to locate a file by starting at the current folder on the current drive, for example, **Support\Validation**. That folder can be located along that relative path only if it exists in the current directory.

Long File Names

Windows operating systems that are 32 bit support long file names. Long file names enable end users to give directories and files more meaningful names. The term *long file name* refers to both long file names and long paths.

InstallShield provides the long file name functions to facilitate the installation of 16-bit applications and 32-bit applications that do not recognize long file names. It is your responsibility to determine your application's requirements. InstallShield provides the tools to help you install any kind of application.

Long File Names and 16-bit Applications

You can launch 16-bit applications using long file names, but you cannot pass long file names as arguments to 16-bit applications; 16-bit applications require the short file name versions of long file names to function correctly.

If your installation passes a file name to a 16-bit application, you must provide a valid short file name to the application. Long file names will not work.

If your installation writes file names to files such as .ini files, and if 16-bit applications are expected to use the files, you must write short file names that the 16-bit applications can use.

Long File Names and Double Quotation Marks

Under 32-bit operating systems, if you pass a long file name containing one or more space characters to the command line (such as in a DOS shell or in the Command Line setting for an icon), you must enclose the long file name in double quotation marks. This is necessary because the command line recognizes the space character as a delimiter separating a command from other arguments. The double quotation marks convert the long file name to a string literal, allowing the command line to receive it as a single argument.

Under 32-bit operating systems, if you pass a long file name containing one or more space characters to the command line, it must be enclosed in double quotation marks. However, due to the manner in which Windows NT accesses icon files, if a long file name in double quotation marks is used in the Command Line setting for an icon, the default Windows icon may display instead of the application's icon.

To display your product's icon, you can specify the icon's path in the parameter szIconPath of the function

AddFolderIcon when using it to add an icon to a program folder.

Double quotation marks must be removed from long file names before they can be converted to short file names in InstallShield. Refer to the LongPathToQuote and LongPathToShortPath functions.

Long File Name Format

Long file names contain names longer than the conventional 8.3 (eight characters plus a three-digit extension) short file name limit. Long file names allow the use of all the characters used in short file names. In addition, long file names can contain plus signs (+), commas (,), semicolons (;), equal signs (=), left and right square brackets ([ ]), and spaces.

Leading and trailing spaces are ignored. The fully qualified long file name (including null terminating character) can be up to 256 characters long on NTFS file systems and 260 characters long on VFAT file systems.

By default, the operating system creates a short file name for every long file name. The short file name consists of the first six characters of the long file name, a tilde (~), and a number.

Defining Constants Through the Compiler

You can define InstallScript constants on the Compile/Link tab of the Settings dialog box instead of in the script.

The following restrictions apply when defining constants on this tab:

- You can define only numeric constants.
- If you define a constant that you defined with a #define statement in the script, a compile error occurs.
- If you define a constant that you defined as a variable in the script, the value of the constant is lost at run time.

Using Windows Constants in a Script

Some Windows constants are predefined in the **ISRTWindows.h** file that is provided in the *InstallShield Program Files Folder***Script\Isrt\Include** folder. This file is automatically included in your installation when you include **Ifx.h** in your script. You do not need to redefine any constants that are defined in **ISRTWindows.h**; doing so will result in a compiler warning. To determine which constants are predefined, refer to the **ISRTWindows.h** file.

To use constants that are not defined in **ISRTWindows.h**, you must define them (using #define) in the declaration block of your installation script. You cannot simply include the **Windows.h** file that is usually part of a C++ program. The values that you need to assign to the undefined constants can generally be found in an include file that is provided with the appropriate Windows SDK or development tool. (For Microsoft Visual C++, most constants can be found in the **Winuser.h** file, which is located in the *InstallShield Program Files Folder***Script\Resource** folder.)

Coding Long String Literals

To make a very long string literal more easily readable in a script, split the long string into two or more shorter strings, place them on consecutive lines, and concatenate them. In InstallScript, concatenation is performed with the plus sign (+). The example below shows how to split a long string literal across several lines of code in this way:

```
szMonths = "January, February, March, April, " +
           "May, June, July, August, September, " +
           "October, November, December";
```

Absolute Path



Project • *This information applies to InstallScript projects.*

An absolute path includes all of the information necessary to locate a file by starting at the root directory of a specified drive. For example, **C:\Program Files\InstallShield\2016** is the absolute path to the InstallShield folder when installed on drive C.

Building Functions

Typically, functions are declared at the top of the file, and the body is defined at the bottom of the file. After you declare a function prototype, you need to define the function itself in the function block. Each function block contains only one function.

Task ***To build a function:***

1. Start the function body with the keyword **function**.
2. Type the return value data type used in the function prototype, or type **void** if the function prototype specifies a return type of void (which indicates that the function does not return a value). If the function prototype does not specify a return type, it is not necessary to enter one here.
3. Type the function name.
4. To the right of the function name, type the names of the arguments that you are using in parentheses. The arguments must correspond in data type to the parameters that you declared in the declare block.

   Note • *InstallScript functions do not allow you to pass an assignment statement as a parameter. In addition, you cannot use the && or || operators within an argument to a function.*

   Steps 1 and 2 create the function header. Function headers do not end with a semicolon in InstallScript.
5. Declare any local variables that you will use in the function. Then type the keyword **begin** on a line by itself, without punctuation.
6. After the begin line, add whichever statements you need in order to accomplish your particular task.

   You can also use a return statement, particularly if you want to return a specific value from the function. See below for information on returning values from a function.
7. End your function with the keyword **end**.

A sample function block is shown below:

```
function GetPathParts(szFullPath, svDrv, svPath, svName)
    LONG lResult;
begin
    lResult = ParsePath(svDrv, szFullPath, DISK);
    if (lResult = 0) then
        lResult = ParsePath(svPath, szFullPath, DIRECTORY);
    endif;

    if (lResult = 0) then
        lResult = ParsePath(svName, szFullPath, FILENAME);
    endif;

    return lResult;
end;
```

Note • *User-defined functions can return a value with a return statement. Other types of data can be returned in parameters that have been declared with the BYREF operator.*

Calling Functions

All functions—whether user-defined, built-in, Sd, or external—are called in the same way. Type the name of the function, followed by the parameters that you want to pass to the function. For example:

```
MyFunction (MyAge, MyHeight, MyWeight);
```

Note *• InstallScript functions do not allow you to pass an assignment statement as a parameter. In addition, you cannot use the && or || operators within an argument to a function.*

An autosized string variable that is passed by reference to a function is not autosized within the called function. If the function attempts to assign a value whose length is greater than the current size of that parameter, run-time error 401 occurs. To avoid this error, declare strings with a specific size when they are to be passed by reference to a function.

Calling a DLL File Function

There are three rules you must remember when you are calling DLL file functions from your InstallScript installation script:

- The maximum length of the DLL file name is 33 characters; the maximum length of the function name is 63 characters.
- The installation cannot accept a composite parameter (that is, a parameter with a width exceeding 4 bytes) when calling a DLL file. However, a parameter can be a pointer that points to a composite structure.
- A string variable whose address is stored in a pointer variable cannot be changed by passing the pointer to a DLL file function. To allow a DLL file function to change the value of a string variable, pass the variable itself as an argument to the function, after declaring the data type of that argument specifying the BYREF operator.

Task ***To call a DLL file function from your script:***

1. Add the DLL file to your project as a support file if you have not already done so. For more information, see Adding Support Files.
1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. In the **InstallScript** explorer, click the InstallScript file (.rul) that should call the DLL function.
3. At the beginning of the script, prototype the function using the following syntax:

   ```
   prototype [CallingConvention] [ReturnType] DLLName.FunctionName( ParamType1, ParamType2, ... );
   ```

   For example:

   ```
   prototype BOOL MyDLL.MyFunction( INT, INT, INT );
   ```

   You can specify the calling convention to be cdecl or stdcall. If you do not specify a calling convention, InstallShield uses stdcall.

   Note that the DLL file name is case-sensitive; for example, the script compiler treats *`MyDLL.MyFunction`* and *`mydll.MyFunction`* as different functions. When prototyping functions from **User32.dll**, **Gdi32.dll**, or **Kernel32.dll**, you may use the keywords USER, GDI, and KERNEL in place of the DLL file name.

   You can specify the return type to be any InstallScript data type except LIST. If you are declaring a DLL file function call in which a wide-character string argument or a Unicode string argument is expected, use the wstring data type. If you do not specify a return type, the installation assumes that the DLL file function returns a 4-byte value.

4. Load the DLL file by calling the **UseDLL** function. For example:

```
UseDLL( SUPPORTDIR ^ "MyDLL.dll" );
```

***Note** • You do not need to load **_isuser.dll**, **_isres.dll**, or Windows API DLL files, such as **User32.dll**, **Gdi32.dll**, and **Kernel32.dll**. Do not call **UseDLL** and **UnUseDLL** to load and unload these DLL files.*

5. Call the function as you would call any other function. For example:

   ```
   bResult = MyDLL.MyFunction( nInt1, nInt2, nInt3 );
   ```

   It is recommended that you include the DLL file name in the function call, as in the preceding example. If your script declares functions with the same name from different DLL files, calling the function without including the DLL file name results in a compiler error.

   If the installation does not find the called function in the DLL file, the installation raises an exception, which you can handle by using a try...catch...endcatch block.

6. After all script calls to the DLL have been made, unload the DLL file by calling **UnUseDLL**. For example:

   ```
   UnUseDLL( SUPPORTDIR ^ "MyDLL.dll" );
   ```

***Note** • You can use the IS_NULLSTR_PTR variable to pass a null pointer to an external DLL function or Windows API through a parameter that has been prototyped as an InstallScript string. For more information, see IS_NULLSTR_PTR.*

Passing an Array to a DLL File Function

InstallScript arrays are internally formatted as OLE Automation safe arrays, not native C or C++ arrays. To pass an InstallScript array to a DLL function that expects a C or C++ array, you must place the array data in the expected format by calling GetCArrayFromISArray.

Declaring Functions

The first step in creating a user-defined function is to declare the function. The keyword *prototype* tells the InstallShield script compiler that the line contains a function definition.

Task ***To declare a function:***

1. Type the keyword **prototype**.
2. Type the function's return type. (This step is optional. If you do not enter a return type, it is assumed that the function returns a NUMBER value or no value.) To specify that the function does not return a value, type **void**.
3. On the same line, enter the function name.
4. After the function name, type the data types of the parameters, and enclose them in parentheses and separate them by commas.
5. If there are no parameters, put empty parentheses to the right of the function name.
6. End the line with a semicolon (;).

In the following examples, **FunctionName** is a function containing three parameters. The arguments passed when calling **FunctionName** must be, in order, INT, STRING, and SHORT. **CopyBitmapExample** has no parameters. **FileTransfer** has five parameters—three LONG variables and two STRING variables—and returns a NUMBER value.

```
prototype FunctionName (INT, STRING, SHORT);
prototype CopyBitmapExample( );
prototype NUMBER FileTransfer (LONG, LONG, LONG, STRING, STRING);
```

When you are declaring DLL functions, use the format <DLL file name>.<function name> for the name of the DLL file function. For example:

```
prototype MyDLL.MyFunction(INT, INT);
```

The above declaration signals to the InstallScript compiler that the program will call a function named **MyFunction**, with two INT parameters, in a file named **Mydll.dll**.

You can also optionally declare a calling convention, either cdecl or stdcall, when declaring a DLL file function. For example:

```
prototype cdecl MyDLL.MyFunction(INT, INT);
```

If you do not explicitly declare a calling convention, InstallShield uses stdcall. If you explicitly declare both a calling convention and a return type, place the calling convention before the return type.



Note *• Most Windows API functions use the stdcall calling convention, but some C or C++ development environments build DLL file functions with the cdecl calling convention unless you prototype your C or C++ function with the __stdcall modifier. For more information, consult your compiler documentation.*

Many Windows API functions are declared in the header file **ISRTWindows.h***, which is automatically included when you include* **Ifx.h** *in your script. (You can prevent the automatic definition of Windows APIs by placing the preprocessor constant ISINCLUDE_NO_WINAPI_H in the Preprocessor Defines box on the Compile/Link tab of the Settings dialog box.)*

Returning Values from Functions

Like InstallScript's built-in functions, user-defined functions can be designed to return a value to the caller. To return a value from a function, you must include a return statement, followed by the value to be returned, before the function's end statement. If you do not include a return statement or if you do not specify a value after the keyword return, the value returned by the function is unpredictable. (If the function prototype specifies a return type of void, the function cannot return a value.)

Many programmers use return statements to return error codes that indicate the success or failure of a function call. Most of InstallScript's built-in functions use a return statement for that purpose. The return statement also is used commonly to create functions that return the result of an operation performed on parameters passed to the function, as in the example below, which returns the area of a rectangle:

```
function RectangleArea (nLength, nWidth)
    INT nVal;
begin
    nVal = (nLength * nWidth);
    return nVal;
end;
```

The keyword return can be followed by a constant, a variable, a numeric or string expression, or a function call. In the example below, **RectangleArea** has been modified to eliminate the assignment statement; the arithmetic expression follows the keyword return:

```
function RectangleArea (nLength, nWidth)
begin
    return (nLength * nWidth);
end;
```

The value returned by a function can be ignored by the calling program or function, tested in a conditional expression, or assigned to a variable. In the following example, the return value from **RectangleArea** is assigned to the variable *nArea*:

```
nArea = RectangleArea (nLong, nWide);
```

In the next example, the result of **RectangleArea** is tested in a conditional expression:

```
if (RectangleArea(nLong, nWide) > nMaxArea) then
        MessageBox("Area exceeds maximum allowed.", INFORMATION);
    endif;
```

To return more than one value or non-numeric values, use the BYREF operator to define parameters that are passed by reference.

Unsupported Functions

Some of the functions that were available in InstallShield Professional and InstallShield—Windows Installer Edition are not supported in InstallShield.

Component Functions

In InstallShield, features are the top-most level of project organization from the end user's perspective. Because of this, component functions are now feature functions. For example, **ComponentDialog** is now **FeatureDialog**. Nearly all of the component functions are available to you as feature functions in InstallShield.



Project • *Feature functions are available for use in InstallScript and InstallScript MSI project types. If you have a Basic MSI project, you must convert your project to the InstallScript MSI project type in order to use feature functions in your installation.*

PreShowComponentDlg and PostShowComponentDlg

In InstallShield—Windows Installer Edition 2.x, you had to call the **PreShowComponentDlg** and **PostShowComponentDlg** functions in order to use component-related (now feature-related) functions. In InstallShield, feature-related functions are available only for InstallScript and InstallScript MSI project types. If you have a Basic MSI project, you must convert your project to the InstallScript MSI project type in order to use feature functions in your script.

Because feature functions are not supported for use in Basic MSI projects, the **PreShowComponentDlg** and **PostShowComponentDlg** functions are no longer necessary and are not supported.

Writing Entry-Point Functions



***Project** • This information applies to the following project types:*

- *Basic MSI with InstallScript custom actions*
- *InstallScript MSI with InstallScript custom actions*
- *Merge Module with InstallScript custom actions*

When Windows Installer executes an InstallScript custom action, InstallShield calls the function that you specified when you created the custom action. Every InstallScript custom action must have an exported, user-defined function as an entry point into the script.

Prototyping and Defining Functions

An entry-point function is prototyped and defined like any other function, except that it has the following requirements:

- Its prototype must include the keyword export to declare it as an exported function.
- The function can accept only one argument, which must be a handle to the .msi database.
- It should return a value meaningful to Windows Installer, if your custom action is designed to wait for a return value. These custom action return values are defined in **IsMsiQuery.h** and available to you if you include **IsMsiQuery.h** or **Iswi.h** in your script.

The following example script declares an entry-point function, which returns success if it succeeded:

```
// Include Isrt.h for built-in InstallScript function prototypes.
#include "isrt.h"

// Include Iswi.h for Windows Installer API function prototypes and constants.
#include "iswi.h"

export prototype MyFunction(HWND);

function MyFunction(hMSI)
    STRING szKey, svValue, svPath;
    NUMBER nvType, nvSize, nReturn;
begin
    RegDBSetDefaultRoot (HKEY_LOCAL_MACHINE);

    szKey = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\isdev.exe";
    nReturn = RegDBGetKeyValueEx (szKey, "Path", nvType, svValue, nvSize);

    // The App Paths key contains the folder where InstallShield was
    // installed, followed by a semicolon.
    StrSub (svPath, svValue, 0, StrFind(svValue, ";"));

    if nReturn = 0 then
        MessageBox ("InstallShield is installed to " + svPath, INFORMATION);
        return ERROR_SUCCESS;
    else
        MessageBox ("Cannot determine where InstallShield is installed.", SEVERE);
        return ERROR_INSTALL_FAILURE;
    endif;
end;
```

Multiple Entry Points

Your script can have multiple entry-point functions in it to serve multiple custom actions. However, InstallShield calls only the single entry-point function specified for each InstallScript custom action.

Adding Function Calls with the Function Wizard

You can use the Function Wizard to select a function, specify its arguments, and insert the function in your script.

Declared Windows API Functions



Project • *This information applies to InstallScript projects.*

Windows API functions are declared in the header file **ISRT.h**, which is automatically included in your installation when you include **Ifx.h** in your script. If this function is also explicitly declared in your script code, do one of the following:

- Remove your function declaration and use the declaration that is provided by InstallShield.
- If you are creating a script that needs to be compatible in both InstallShield 2016 and versions of InstallShield Professional earlier than 7, surround your declaration with code such as the following:

```
#if _ISCRIPT_VER < 0x700prototype ...#endif
```

You may also need to update code that calls the Windows API function, if the declaration that is defined by InstallShield is different than your declaration.

You can prevent the automatic definition of Windows APIs by placing the preprocessor constant ISINCLUDE_NO_WINAPI_H in the Preprocessor Defines box on the Compile/Link tab of the Settings dialog box.

Specifying a Non-Default Feature Event Handler Function

By default, if the file **Setup.rul** exists in the InstallScript view and defines a function <feature name>_<event name> (for example, **NewFeature1_Installing**), then that function is the handler for that event of that feature.

You can select a different function to be the handler for the feature event. It can be any function defined in any script file in the InstallScript view, as long as it takes no arguments and is declared using the export keyword, for example:

```
export prototype MyFeatureHandler();
```



Task ***To specify a non-default feature event handler function, do the following:***

1. Open a script file (.rul file) in the **InstallScript** view.
2. Do one of the following in the script editor pane:
   - Declare and define the function that you want to specify as a feature event handler.
   - In the Event Category and Event lists at the top of the script editor pane, select the feature and select an event handler. InstallShield automatically creates the event handler (for example, Feature1Installing).



Note *• If you put parentheses in a feature's name, event handler functions with names of the default form <feature name>_<event name> will not compile properly.*

Accessing Global Variables



Project *• This information applies to InstallScript MSI projects.*

A global variable is data declared outside of a function and available to any module in the script. By using global variables, you can share persistent data with event handlers, entry-point functions, and user-defined functions.

For example, if you retrieve the value of the operating system's version in your OnBegin event handler, you can later access that global value in an entry-point function. The following script excerpt illustrates this:

```
// Include header file for built-in functions
#include <isrt.h>
// Include header file for event handlers and MSI APIs
#include <iswi.h>

    // Declare global variable
    NUMBER nvResult;

    // Prototype entry-point function
    export prototype MyFn(HWND);

function OnBegin()
    // Declare local variable
    STRING svResult;
begin
    GetSystemInfo (WINMAJOR, nvResult, svResult);
end;

function MyFn(hMSI)
begin
    if nvResult > 4 then
        // Code for version of Windows later than 4.0
    else
        // Code for Windows 4.0
    endif;
end;
```



Important *• Global variables share state across all event handlers. However, if you create an InstallScript custom action for your project, global variables will not share state from the InstallScript custom action to the event-driven InstallScript code, or vice versa. Thus, if you declare a global variable in your event-driven InstallScript code, that variable would not be available to your InstallScript custom action. Similarly, if you declare a variable in your InstallScript custom action, that variable would not be available to your event-driven InstallScript code.*

Uninstalling Initialization (.ini) File Entries

The uninstallation removes modifications that are made with the InstallScript initialization file functions while logging is enabled. For more details, see the information in this section of the documentation.

General Limitations of Uninstalling Initialization (.ini) File Entries

The following limitations apply to uninstalling .ini file entries and modifications automatically:

- Logging must be enabled when the .ini file functions are called.
- If an .ini file was created by any of the .ini file functions, it is not removed.
- If a section name was created by any of the .ini file functions, it is not removed, even if no more keys are present.
- If a key value is completely replaced by an InstallScript function (not appended to, or prepended to), the key values that existed before the installation are not restored. In this case InstallShield considers the replaced key a newly created key and will uninstall it as if it is a new key that was created by the installation.
- The uninstallation does not restore keys or values that were deleted using **WriteProfString**.
- To append a value to existing key (for example, network under [386Enh]), the new entry must either be appended at the end or prepended in the very beginning of the existing string, but never inserted in the middle.
- Only the comma (,) and semicolon(;) are considered valid delimiters. If a string value to be uninstalled is found as a part of longer string, the value and the delimiter before or after it is also removed appropriately (based on the string's position) only if the delimiter is a comma or a semicolon. For example, if the uninstallation removes **pqr** from the string **Key=pqr,rst,uvw**, it also removes the comma after **pqr**. A character other than a comma or a semicolon in its place will not be removed. As a rule, avoid adding spaces around delimiters.

Uninstalling AddProfString's Initialization (.ini) File Entries

The uninstallation removes the keyname and value pair completely if the following conditions are true:

- The key was successfully created by **AddProfString**.
- The keyname and the value pair that exist when the uninstallation is run exactly match the installed keyname and value pair. If another installation or program modifies the installed key between the installation and uninstallation, the keyname and value pair will not be removed completely. Only the value that was created by the original installation will be removed; the key itself and any additional values will not be removed by the uninstaller.

For example, if the **System.ini** file originally read as follows:

```
[386Enh]
device=votc.386
device=*vmpcd
```

and after your call to AddProfString, it read as follows:

```
[386Enh]
device=*vmp32
```

```
device=votc.386

device=*vmpcd
```

the uninstallation would remove `device=*vmp32`.

If another installation had added a value to the existing line, only the value from the installation that is being uninstalled will be removed. The original keyname and the new value will be left in the file. For example, if you had added the line **Test Values=votc.386** in **Test.ini** under the `[Test]` section:

```
[Test]

Test Values=votc.386

Continuous Test=No
```

and another installation had added a new value to Test Values after your installation had run:

```
[Test]

Test Values=votc.386,pctcp.386

Continuous Test=No
```

when your application is uninstalled, `votc.386` will be removed and `TestValues=pctcp.386` will be left in the file:

```
[Test]

Test Values=pctcp.386

Continuous Test=No
```

Uninstalling ReplaceProfString's Initialization (.ini) File Entries

The uninstallation removes the keyname and value pair completely if the following conditions are true:

- The key was successfully created by **ReplaceProfString**.
- The key did not previously exist.
- The keyname and value pair that exist when the uninstallation is run exactly match the installed keyname and value pair. If another installation or program modifies the installed key between the installation and uninstallation, the keyname and value pair will not be removed completely. Only the value that was created by the original installation will be removed; the key itself and any additional values will not be removed by the uninstaller.

For example, if the **System.ini** file originally read:

```
[386Enh]

device=votc.386

device=*vmpcd
```

and after your call to **ReplaceProfString**, it read:

```
[386Enh]

device=*vmp32

device=votc.386

device=*vmpcd
```

the uninstallation would remove `device=*vmp32`.

Uninstalling WriteProfString's Initialization (.ini) File Entries

The uninstallation removes the keyname and value pair completely if the following conditions are true:

- The key was successfully created by **WriteProfString**.
- The key did not previously exist.
- The keyname and value pair that exist when the uninstallation is run exactly match the installed keyname and value pair. If another installation or program modifies the installed key between the installation and uninstallation, the keyname and value pair will not be removed completely. Only the value that was created by the original installation will be removed; the key itself and any additional values will not be removed by the uninstaller.

Extending Your Installation with COM Objects



Important • *Do not confuse COM objects with InstallShield objects. They are two separate features.*

Instead of using DLLs or other executable files, you can extend your installation with COM objects. COM objects can be integrated into your installation fairly easily. Use the following procedure to assign COM objects in your script.



Task ***To assign a COM object:***

1. Declare an object variable. For example:

   ```
   OBJECT oMyCOMObject
   ```

2. Get a reference to your COM object and assign it to the variable by using the set keyword. For example:

   ```
   set oMyCOMObject = CreateObject ( szMyProgID );
   ```

The value of szMyProgID is a valid program ID for your COM object. If you leave the set keyword out, the script engine attempts to set the default property of oMyCOMObject to szProgID. Because there is no default property, since oMyCOMObject does not contain a reference to your COM object yet, the script will fail.



Important • *By default, once a COM object is loaded by the installation using* ***CoCreateObject****,* ***CoCreateObjectDotNet****,* ***CoGetObject****, or* ***DotNetCoCreateObject****, it remains loaded until the installation is finished. The COM object remains loaded regardless of whether the COM object variable goes out of scope or is set to NOTHING.*

Consequently, the library referenced by the COM object is also not released until the installation is finished. However, it is possible to unload the library referenced by the COM object by calling the Windows API ***CoFreeLibrary****.* ***CoFreeLibrary*** *unloads the library regardless of whether any object variables still reference it. Therefore, you should only call* ***CoFreeLibrary*** *after all objects that reference the library have gone out of scope or are set to NOTHING. See the following example code:*

```
prototype void ole32.CoFreeLibrary( byval int );
int hModule;

program

    // Free the library.
```

```
    hModule = GetModuleHandle( "CSharpClassLibrary.dll" );

    if( !hModule ) then
       MessageBox( "Failed to get module handle", INFORMATION );

    else

       CoFreeLibrary( hModule ); // Release the library

    endif;

endprogram
```

CoFreeUnusedLibraries *can also be used to unload the library. However, by default, the system waits 10 minutes before unloading the library, and in most cases, the installation requires the library to be unloaded immediately. If your installation will run on Windows XP and later only, you can call* ***CoFreeUnusedLibrariesEx****, which includes the option of unloading the library immediately.*

Below is an example COM object that validates a serial number. It has one method, Validate, and one property, IsEval.

```
function OnFirstUIBefore()
    OBJECT oObj;
    STRING szProgID, szMsg, szTitle, svName, svCompany, svSerial;
    BOOL bValidSerialNumber, bEval, bValidate;
    NUMBER nResult;
begin
    szProgID = "MySerialValidation";
    set oObj = CreateObject( szProgID );
    if ( !IsObject( oObj ) ) then
        MessageBox( "Object " + szProgID + " is invalid!", SEVERE );
        return ISERR_GEN_FAILURE;
    endif;

    bValidSerialNumber = FALSE;
    while (!bValidSerialNumber)
        szMsg = "Please enter your name, company name, and product serial number.";
        szTitle = "Enter info";
        nResult = SdRegisterUserEx( szTitle, szMsg, svName, svCompany, svSerial );
        if ( nResult < 0 ) then
            MessageBox( "Failed to display dialog box.", INFORMATION );
            return ISERR_GEN_FAILURE;
        endif;

        /* Check the value of the object's IsEval property. */
        bEval = oObj.IsEval;
        /* Execute the object's Validate method and
        get the return value from the method. */
        bValidate = oObj.Validate( svSerial );
        if ( bEval || bValidate ) then
            bValidSerialNumber = TRUE;
        else
            MessageBox( "Invalid serial number; please re-enter.", INFORMATION );
        endif;
    endwhile;

    return ISERR_SUCCESS;
end;
```



Note *• You can pass a data structure to your COM object as long as it has the members that your COM object expects. The following is an example of passing a data structure:*

```
#define PERSON_NAME_SIZE 1024

typedef PERSON
    begin
        STRING Name[PERSON_NAME_SIZE];
        NUMBER Age;
    end;

function OnBegin()
    PERSON pPerson;
    NUMBER nPhoneNumber;
    OBJECT oMyCOMObject;
    STRING szMyProgID;
begin
    /* Assign a value to szMyProgID in this line. */
    set oMyCOMObject = CreateObject ( szMyProgID );
    if ( !IsObject( oMyCOMObject ) ) then
        MessageBox( "Object " + szMyProgID + " is invalid!", SEVERE );
        return ISERR_GEN_FAILURE;
    endif;
    nPhoneNumber = oMyCOMObject.GetPhoneNumber( pPerson );

    return ISERR_SUCCESS;
end;
```



Note *• The standard COM return value HRESULT is not visible to an automation client such as InstallScript. To return a value from your COM object method, you should have an [out,retval] parameter in your method's parameter list, as illustrated in the following code samples.*

IDL:

```
interface IMyInterface : IDispatch
{
    [id(1)] HRESULT DoSomeWork([in] long nInputValue, [out,retval] long *pReturnValue);
}
```

C++:

```
STDMETHODIMP MyInterfaceImpl::DoSomeWork(long nInputValue, long *pReturnValue)
{
    // your implementation code
    *pReturnValue = 1; //return a value to client
    return S_OK;
}
```

InstallScript:

```
function OnBegin()
    NUMBER nObjReturn;
begin
    set oMyCOMObject = CreateObject("MyProgID");
    if (IsObject(oMyCOMObject) then
        nObjReturn = oMyCOMObject.DoSomething(10);
```

```
        /*nObjReturn will contain the value 1, returned from your implementation*/
    endif;
end;
```

Specifying Features and Subfeatures in Function Calls



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Feature is a general term that refers to a set of components or subfeatures in InstallShield. A *subfeature* is a feature that is located below another feature—similar to the relationship between a folder and a subfolder.

Top-level features are the highest features in the hierarchy. Top-level features are never referred to as subfeatures.

How to Refer to Features and Subfeatures in InstallScript Code

Some feature functions and setup type dialog functions require you to refer to a single feature, while others require you to refer to multiple features.

Referring to Single Features

To refer to a single feature, use the feature's name. To refer to a subfeature, use a path-like expression where the name of each feature in the hierarchy leading to that feature is separated by double backslashes. For example, to specify the subfeature **Tutorials** under the top-level feature **Help Files**, use the following expression in your installation script:

```
szFeature = "Help Files\\Tutorials";
```

To refer to the subfeature **CBT** under **Tutorials**, use the following:

```
szFeature = "Help Files\\Tutorials\\CBT";
```

Note that the name of a feature cannot contain backslashes.

Referring to Multiple Features

In InstallScript MSI installations, some feature and setup type dialog functions, such as **SdFeatureMult**, display multiple features and their subfeatures. In these cases, you refer to multiple features by specifying the feature immediately above them in the hierarchy. To refer to multiple top-level features, use a null string ("").

For example, if you pass a null string to the **SdFeatureMult** function, the corresponding dialog displays all the top-level features in your script-created component set in the left window on the **SdFeatureMult** dialog, depending on the value of the MEDIA system variable. All subfeatures appear in the right window on this dialog.

Calling a Windows API Function



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

You can call a Windows API function from within your InstallScript code.

Task ***To call a Windows API function:***

1. At the beginning of the script, prototype the function using the following syntax:

   ```
   prototype ReturnType DLLName.FunctionName( ParamType1, ParamType2, ... )
   ```

   For example:

   ```
   prototype INT User.LoadString( INT, SHORT, BYREF WSTRING, INT );
   ```

   Use **Dumpbin.exe** with the `/EXPORTS` option to determine which functions reside in any of the Windows API DLLs **Kernel32.dll**, **User32.dll**, and **GDI32.dll**. To determine the return type and parameter types for a function, consult a Windows programming reference such as the Microsoft Development Library (MSDL).

   Note • *You can specify the return type to be one of the following InstallScript data types: BOOL, CHAR, HWND, INT, LONG, LPSTR, NUMBER, POINTER, or SHORT. If a return type is not specified, InstallShield assumes that the DLL function returns a 4-byte value.*

2. Call the function as you would call any other function. For example:

   ```
   nResult = LoadString( iInstance, nStringID, szMyString, MAX_SIZE );
   ```

3. To get extended error information if the function fails, check the value of the Err object's LastDllError property. (It is not possible to call the Windows API function **GetLastError** to get extended error information, since the setup itself changes this value before returning control to the script.)

Embedding Custom Transfer File Operations

In InstallShield, you can embed additional file transfer operations during the standard file transfer of the installation. Also, the progress bar is updated smoothly and appropriately during these operations. There are a few different common scenarios in which you would use this functionality.

This table provides an overview of what functions are called for each scenario.

Table 4-3 • Determining Which Functions Are Appropriate for Common Custom Transfer File Operations

Scenario Using	Function that Should Be Called
FeatureMoveData	None
XCopyFile or CopyFile	FeatureAddCost
LaunchApplication (external applications)	FeatureAddCost, FeatureSpendCost, FeatureAddUninstallCost, or FeatureSpendUninstallCost

These procedures include step-by-step information about how to use custom file transfer operations for each scenario.



Task ***To install additional files during the standard file transfer operation using XCopyFile or CopyFile:***

1. Before file transfer, determine the size of the files to be installed, taking into account the cluster size on the target folder for the files. You can accomplish this using the **GetAndAddFileCost**, **CalculateAndAddFileCost**, and/or **GetAndAddAllFilesCost** functions.
2. Using the **FeatureAddCost** function, add the cost determined in Step 1 to the cost for the feature that these files are associated with. Note that all cost must be associated with a particular feature. Therefore, if the files to be installed are not associated with an existing feature, you must create a dummy feature or use an existing feature.
3. During file transfer, call the **XCopyFile** function to install the files. Typically, this would be done during the Installing event of the feature associated with the additional files or in the OnMoving or OnMoved events. The progress bar is updated smoothly and appropriately during the call. Note that you should disable the Cancel button in the status dialog during the **XCopyFile** operation. See the description for the XCopyFile function for more information.



Note • *There is no additional action required for the uninstallation. The files are uninstalled while the uninstallation progress is updated appropriately.*



Task ***To call an external installation that installs additional files during the standard file transfer operation:***

1. Before file transfer, determine the size and number of the files to be installed (or other operations) by the external installation by examining the external installation and taking note of what operations occur.
2. Using the **FeatureAddCost** function, add the cost determined in Step 1 to the cost for the feature that this installation is associated with. Note that all cost must be associated with a particular feature. Therefore, if the files to be installed are not associated with an existing feature, you must create a dummy feature or use an existing feature. Review the two scenarios below.
3. If the external installation is called during uninstallation, add the number of operations and size of the operations to the uninstallation cost using **FeatureAddUninstallCost**.
4. During file transfer, use the **LaunchApplication** event to launch the installation, typically in silent mode. Typically, this would be done during the Installing event of the feature associated with the additional files or in the OnMoving or OnMoved events. Use the LAAW_USE_CALLBACK option so that your script regains control periodically during the installation and uninstallation.
5. Override the **OnLaunchAppAndWaitCallback** event, and add code to poll the running installation to determine its progress. Then call the **FeatureSpendCost** (Installation) function or the **FeatureSpendUninstallCost** (Uninstallation) function to spend the appropriate amount of cost that the installation has completed. Repeat this process while the external installation is running. The progress bar is updated smoothly and appropriately during the call. Note that you might have to adjust the LAAW_PARAMETERS.nCallbackInterval so that the **OnLaunchAppAndWaitCallback** event is called often enough to keep the progress updated smoothly.

Expressing Large Numbers in InstallScript

When a number is stored in an InstallScript integer, only 31 bits of data are available to store the value. Therefore, the maximum value that can be expressed is 2^31 (2 GB). Note also that since InstallScript integers are always considered to be signed, the 32 bit is interpreted as the sign bit and cannot be used to store numeric data. This limits the values that can be expressed to 2^31 (2 GB) instead of 2^32 (4 GB), which could be stored in an unsigned data type.

In some cases, it is necessary or desirable to be able to specify larger values. To accomplish this in InstallShield, a number of functions support the specification of large numbers as a pair of 32-bit numbers with the sign bit (32 bit) of each number always being set to 0 to indicate a positive number. Using this format, it is possible to specify and retrieve a number with up to 62 bits of data or 2^62 of size.

Note however that since the language does not have a data type that can express the number as a single variable, the operations that can be performed on this data are limited. This data is typically passed between functions that support these values. The data can also be converted to a single value expressed in a larger size unit (such as KB or MB) using the **ConvertSizeToUnits** function or passed to an external DLL function that can handle larger data types.

When a number is specified as a high/low pair, the lower 32-bit value specifies the lower 31 bits of data (with the sign bit always set to 0 and ignored) of the full value. The upper 32-bit value specifies the upper 31 bits of data (again with the sign bit always set to 0 and ignored). For example, an upper value of 4 actually indicates 4 * 2^31 or 8589934592 (8 GB) of size. This, combined with a lower value of 100, would indicate a total size of 8589934692 units.

Note that in the case of numbers less than 2^31, the high value will always be 0 and thus can be ignored if you are sure that the size of the value will never exceed 2^31. However, it is recommended that if you need to use this data for computations, you use the **ConvertSizeToUnits** function to convert this data to a single value of the most appropriate units with minimal amount of rounding to express the data in a single value.



***Note** • Note that convention differs from the unsigned high/low value pairs used by Windows in which the lower value stores the lower 32 bits of data and the high value stores the upper 32 bits of data. The **ConvertWinHighLowSizeToISHighLowSize** function is provided for easy conversion of data returned by Windows API functions.*

The following functions support high/low value pairs:

- GetDiskInfo
- GetFileInfo
- GetAndAddFileCost
- CalculateAndAddFileCost
- GetAndAddAllFilesCost
- FeatureFileInfo
- FeatureGetCostEx
- FeatureAddCost
- FeatureSpendCost
- ConvertWinHighLowSizeToISHighLowSize
- ConvertSizeToUnits

Installing Device Drivers

InstallShield supports installing device drivers in InstallScript installations using Windows Driver Install Frameworks (DIFx), which includes the DIFxAPI component. This component allows drivers to be installed or uninstalled without using the Windows Installer.

Project • *This information applies to InstallScript projects. Installing device drivers is supported in InstallScript MSI and Basic MSI projects using the DIFx Windows Installer functionality. For information on configuring device drivers using the Windows Installer, see Configuring Device Driver Settings.*

Task ***To install a 32-bit DIFx driver in an InstallScript project:***

1. In the View List under **Installation Information**, click **General Information**.
2. For the **DIFx Support (for 32-bit platforms)** setting, select **Enabled**.
3. Add the DIFx driver files to the project.
4. Override the feature event for the feature containing the DIFx driver files and either call the **DIFxDriverPackageInstall** function for non-Plug and Play (PnP) drivers or the **DIFxDriverPackagePreinstall** function for PnP drivers.
5. Build and run the installation.

Task ***To install a 64-bit DIFx driver in an InstallScript project:***

1. In the View List under **Installation Information**, click **General Information**.
2. For the **DIFx Support (for 64-bit Itanium platforms)** setting or the **DIFx Support (for 64-bit AMD platforms)** setting, select **Enabled**.
3. Add the DIFx driver files to the project.
4. Override the feature event for the feature containing the DIFx driver files and either call the **DIFxDriverPackageInstall** function for non-Plug and Play (PnP) drivers or the **DIFxDriverPackagePreinstall** function for PnP drivers.
5. Build and run the installation.

Checking the Compiler Version

In InstallShield, _ISCRIPT_VER is defined as 0x*VVM*, where *VV* is the major version of the product and *M* is the maintenance pack release number. For example, 0x900 evaluates as InstallShield DevStudio version 9.0 and maintenance pack 0 (the first release). This preprocessor constant was defined in InstallShield Professional 7 as 0x700 and in InstallShield Professional 6 as 0x600, but is undefined and evaluates as zero in InstallShield Professional 5.x.

You can use this constant to test whether the InstallShield compiler is being used by including code like the following in your script. This code tests for the existence of the first release of InstallShield DevStudio (version 9.0, maintenance pack number 0) or later.

```
#if _ISCRIPT_VER >= 0x900
    MessageBox( "DevStudio 9.x and above",INFORMATION );
#elif _ISCRIPT_VER >= 0x700
    MessageBox ( "This is an IS 7.x compiler.", INFORMATION );
#else
    MessageBox ( "This is an IS 6.x compiler or below.", INFORMATION );
#endif
```

Preprocessor Constants Maintained for Backwards Compatibility

The preprocessor constants _ISCRIPT_ISPRO and _ISCRIPT_ISDEV will be maintained for backwards compatibility. _ISCRIPT_ISPRO is defined for InstallScript projects (and InstallShield Professional projects). _ISCRIPT_ISDEV is defined for InstallScript MSI projects and Basic MSI projects.

Checking the Authoring Environment with _ISCRIPT_ISDEV and _ISCRIPT_ISPRO



Project • *This information applies to InstallScript projects.*

The preprocessor constant _ISCRIPT_ISPRO is defined in InstallScript projects (and in InstallShield Professional projects) but is undefined and evaluates as zero in InstallScript MSI and Basic MSI projects. The preprocessor constant _ISCRIPT_ISDEV is defined in InstallScript MSI and Basic MSI projects but is undefined and evaluates as zero in InstallScript projects (and in InstallShield Professional projects).

You can use _ISCRIPT_ISDEV and _ISCRIPT_ISPRO to write a single script that produces different behavior in the different project types by including code like the following in your script:

```
#ifdef _ISCRIPT_ISPRO
    // code specific to InstallShield Professional and InstallScript projects
#else
    #ifdef _ISCRIPT_ISDEV
        // code specific to Basic MSI and InstallScript MSI projects
    #endif
#endif
```

Launching an Installation from Another InstallScript Installation



Project • *The following information applies to InstallScript projects.*

You can launch a complete installation by calling the **LaunchApplication** function. For example, if you placed the child installation's files on a CD-ROM, you could call **LaunchApplication** as follows:

```
LaunchApplication (SRCDISK ^ "Launched Setup Folder\\Setup.exe", "", "", SW_HIDE,"", LAAW_OPTION_WAIT);
```

As an alternative, you can include the child installation in your project as a support file. If you do this, your main installation copies the child installation to the target system, runs the installation, and deletes the installation along with any other support files. Use the SUPPORTDIR variable in the path of your child installation:

```
LaunchApplication (SUPPORTDIR ^ Setup.exe, "", "", SW_HIDE,"", LAAW_OPTION_WAIT);
```

Alternatively, you could insert links to the child installation into your file groups, install the child installation onto the target system, and then launch the installation from the target location. (With these methods, the child installation is left on the target system until the parent installation is uninstalled.)

Language Support for InstallScript

The table below shows the languages supported by the Premier edition of InstallShield. Following are descriptions of each of the columns:

- **InstallShield Language**—Name used by the InstallShield interface to refer to this language.
- **InstallScript Constant**—Language constant provided by InstallShield for filtering language-specific components.
- **Windows English Equivalent**—Name that the English version of Windows uses to refer to the language.

Table 4-4 • Supported Languages

InstallShield Language	InstallScript Constant	English Equivalent
Basque	ISLANG_BASQUE	Basque
Bulgarian	ISLANG_BULGARIAN	Bulgarian
Catalan	ISLANG_CATALAN	Catalan
Chinese (Simplified)	ISLANG_CHINESE_SIMPLIFIED	Chinese (Simplified)
Chinese (Traditional)	ISLANG_CHINESE_TRADITIONAL	Chinese (Traditional)
Croatian	ISLANG_CROATIAN Note that for backward compatibility, this constant continues to be 0x001a rather than the more logical 0x041a (in light of its relation to Serbian); you should continue to use this constant rather than ISLANG_CROATIAN_STANDARD.	Croatian
Czech	ISLANG_CZECH	Czech
Danish	ISLANG_DANISH	Danish
Dutch	ISLANG_DUTCH	Dutch (Standard)
English	ISLANG_ENGLISH_UNITEDSTATES	English (United States)
Finnish	ISLANG_FINNISH	Finnish
French (Canadian)	ISLANG_FRENCH_CANADIAN	French (Canadian)

Table 4-4 • Supported Languages (cont.)

InstallShield Language	InstallScript Constant	English Equivalent
French (Standard)	ISLANG_FRENCH_STANDARD	French (Standard)
German	ISLANG_GERMAN	German (Standard)
Greek	ISLANG_GREEK	Greek
Hungarian	ISLANG_HUNGARIAN	Hungarian
Indonesian	ISLANG_INDONESIAN	Indonesian
Italian	ISLANG_ITALIAN	Italian (Standard)
Japanese	ISLANG_JAPANESE	Japanese
Korean	ISLANG_KOREAN	Korean
Norwegian	ISLANG_NORWEGIAN	Norwegian (Bokmal)
Polish	ISLANG_POLISH	Polish
Portuguese (Brazilian)	ISLANG_PORTUGUESE_ BRAZILIAN	Portuguese (Brazilian)
Portuguese (Standard)	ISLANG_PORTUGUESE_STANDARD	Portuguese (Standard)
Romanian	ISLANG_ROMANIAN	Romanian
Russian	ISLANG_RUSSIAN	Russian
Serbian	ISLANG_SERBIAN_CYRILLIC	Serbian (Cyrillic)
Slovak	ISLANG_SLOVAK	Slovak
Slovenian	ISLANG_SLOVENIAN	Slovene
Spanish	ISLANG_SPANISH	Spanish (Traditional Sort)
Swedish	ISLANG_SWEDISH	Swedish
Thai	ISLANG_THAI	Thai
Turkish	ISLANG_TURKISH	Turkish

Preventing Color Distortion

If you are experiencing problems with color distortion in your InstallScript custom action, it is most likely due to shifts in the color palette. This section explains how colors are apportioned on Windows systems and offers troubleshooting tips to help you minimize any distortion in your installation's end-user interface.

On systems operating in 256-color mode, there is one 256-color palette for displaying all available colors; in 16-color mode, there is a 16-color palette, regardless of the number of colors that the video driver supports. Systems operating in high-color or true-color mode do not use a color palette of any type. The colors are displayed directly, so color distortion should not occur on these systems. On systems using a color palette, entries are allocated and used as needed when an object—such as a bitmap—is displayed.

Once the current color palette has been allocated, to display a new color Windows attempts to use a color similar to one that was already allocated. Therefore, distortion may result on a 256-color system if multiple objects that contain several different colors are displayed simultaneously.

Also, Windows allocates 16 static colors (VGA colors) and four additional shades of gray during startup. These colors are used by the system to display 16-color images and are never de-allocated by Windows. See the Windows platform SDK for more information on color palette handling.

Use the tips below to minimize any color distortion related to color palette shifts. Color palette tips apply to all images that you display during installation, including the background color, .avi files, metafiles, and bitmaps.

Main Installation Window Background

- Try using a solid color background, which requires only one palette entry. Call the **SetColor** function with a solid background constant to create a solid background.
- Try using a 16-color gradient background, which requires only 16 palette entries. Call the **SetColor** function with one of the gradient color constants to create this type of background.
- Try using a tiled or a centered bitmap background. Call the **PlaceBitmap** function to create one of these backgrounds.
- Avoid using a 256-color gradient background, which requires about 80 palette entries.

Bitmaps and Metafiles

- When a bitmap is removed by calling the **PlaceBitmap** function with the REMOVE option, all color palette entries used by that bitmap that are not being used by any other currently displayed image are freed from the color palette when other colors are needed to display other objects. If you experience color distortion, always remove a bitmap once it is no longer visible (instead of covering it with another bitmap) so that its color palette entries are freed.
- Try to use similar colors (including the 16 static colors, which are always allocated) for bitmaps and billboards displayed during the installation to reduce the number of color palette entries needed.
- You can determine the maximum colors available on the target system by calling the **GetSystemInfo** function with the COLORS option. Perhaps you will want to display bitmaps of different resolutions based on the return value.
- Note that 24-bit bitmaps do not include a custom color palette in the bitmap file. When displaying a 24-bit bitmap on a 256-color system, only the currently available palette entries will be used, even if there are additional color palette entries available. If you are displaying 24-bit bitmaps in your installation and you expect it to run on 256-color systems, it is recommended that you also include 256-color versions of your bitmaps.
- Since metafiles are drawn instead of placed, additional color palette entries are not allocated when a metafile is drawn; only the existing color palette entries are used to display the metafile. If you expect your installation to run on 256 color systems, it is recommended that you use only the standard 16 static colors in your metafile, because these colors will be available, regardless of the current color palette's availability.

- Verify that the system's video driver supports 256 or more colors. If the video driver does not support 256 colors, the bitmap will not display properly, even if your video card and monitor support 256 colors.
- If a monitor is not capable of displaying 256-color bitmaps, only the 16 static colors are used. This can cause severe distortion problems. If you are using 256-color images in your installation, it is recommended that you require end users to run it on systems that support 256 colors.

Using Custom Actions



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield supports the use of custom actions to run InstallScript, VBScript, or JavaScript code; call DLL functions; run executable files; call a managed method in a managed assembly; set a property or a directory; trigger an error and end the installation; run PowerShell scripts; terminate a process; or run other installation packages.



Task ***InstallShield divides the task of creating and implementing custom actions into the following steps:***

1. Create a custom action in the Custom Actions and Sequences view (or the Custom Actions view) or by using the Custom Action Wizard.
2. Decide when your custom action should run.
3. Launch a custom action by inserting it into a sequence or placing it as the result of a dialog's control event (Basic MSI projects).

For details about each of the InstallShield custom actions that are added automatically to InstallShield projects to support different functionality, see InstallShield Custom Action Reference.



Project *• Merge module projects only: You can control the launch of custom actions in a merge module by modifying the* ***ModuleInstallExecuteSequence*** *table in the Direct Editor. When you add the merge module to your installation project, all custom actions and dialogs included in the merge module are available for you to insert in the installation's sequences via the Custom Actions and Sequences view.*

Using the Custom Action Wizard



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You can use the Custom Action Wizard to create a custom action or modify an existing one.



Task ***To launch the Custom Action Wizard:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in DIM, Merge Module, and MSM Database projects).
2. Right-click the **Custom Actions** explorer and click **Custom Action Wizard**. The Custom Action Wizard launches.
3. Follow the wizard panels to create your custom action.

Alternatively, you can create a custom action in the Custom Actions and Sequences view (or the Custom Actions view) without using the Custom Action Wizard. For more information, see Creating Custom Actions in the Custom Actions and Sequences View (or the Custom Actions View).

To learn about displaying action information on the progress dialog and in the installation's log file, see Using Action Text.

Creating Custom Actions in the Custom Actions and Sequences View (or the Custom Actions View)



Project *• The Custom Actions and Sequences view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The view is called the Custom Actions view in the following project types:

- *DIM*
- *Merge Module*
- *MSM Database*

The recommended way to create a custom action is with the Custom Action Wizard. The wizard walks you through all the required steps to create a custom action and prompts for information. You can also create your custom action in the Custom Actions and Sequences view (or the Custom Actions view)—without using this wizard—by configuring all of its settings directly in the view.

Task ***To add a custom action:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in DIM, Merge Module, and MSM Database projects).
2. Right-click the **Custom Actions** explorer and then click the type of custom action that you want to add.
3. Rename the new custom action by selecting it, pressing the F2 key, and typing the new name.
4. Configure the custom action's settings.

To learn about displaying action information on the progress dialog and in the installation's log file, see Using Action Text.

Cloning Custom Actions

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield provides the ability to copy or *clone* your custom actions. Cloning creates a new custom action of the same type and with all of the same properties and values as the original custom action. You can use cloning to create multiple custom actions that have similar attributes without having to manually set every custom action attribute.

Note that the clone operation clones only the custom action. It does not insert the custom action into any of the installation sequences.

Task ***To clone a custom action:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in DIM, Merge Module, and MSM Database projects).
2. In the **Custom Actions** explorer, right-click the custom action that you want to clone and click **Clone**.

InstallShield adds a copy of the cloned custom action to the Custom Actions explorer. The name of the custom action is the same name as the cloned custom action, except that the copy has a **1** at the end of the name.

Exporting Custom Actions to Other Projects

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Task ***To save a custom action to another project:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI and InstallScript MSI projects) or **Custom Actions** (in DIM and Merge Module projects).
2. In the **Custom Actions** explorer, right-click your custom action and click **Export**. The **Export into** dialog box opens.
3. Browse to the location of an existing InstallShield project file (.ism file). It can be either an installation project or a merge module project, but the file cannot be open in another instance of InstallShield.
4. Click **Open**.

A copy of this custom action is added to the specified .ism file. If the other .ism file already has a custom action of that name with different properties, the Resolve Conflict dialog box opens to offer you options for resolving the conflicts.

Configuring Custom Action Settings

Project *• The Custom Actions and Sequences view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The view is called the Custom Actions view in the following project types:

- *DIM*
- *Merge Module*
- *MSM Database*

When you create a custom action, you need to configure its settings. You may also need to later modify its settings. For example, you might want to edit the condition that is assigned to a custom action.

Depending on the type of custom action that you want to create, the settings for that action may have different meanings. For example, if your action calls an executable file that is stored in the .msi package, the Source setting needs to have the name of the entry in the table that points to the executable file that you want to call. The target setting is a command-line argument in this case.

Task ***To configure the settings for a custom action:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in DIM, Merge Module, and MSM Database projects).
2. In the Custom Actions explorer, click the custom action whose settings you want to configure.
3. In the right pane, configure the settings as appropriate.

For information about each setting, see Custom Action Settings.

To learn about displaying action information on the progress dialog and in the installation's log file, see Using Action Text.

Custom Action Type Overview

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The value that is displayed in the MSI Type Number setting for a custom action represents a combination of the type of custom action (such as standard DLL, executable file, or InstallScript), the location (such as stored in the **Binary** table or installed with the product) as well as several of the custom action's settings, such as Return Processing, In-Script Execution, and Execution Scheduling. It is the numeric value that is calculated by using the OR operator to combine several bits that are available for the Type column in the Windows Installer **CustomAction** table. InstallShield also offers support for extended custom action types, such as InstallScript custom actions and standard DLL file functions.

The Windows Installer Help Library refers to custom actions by a numeric type. When you add a custom action to your project in the Custom Actions and Sequences view (or the Custom Actions view), the custom action type number is calculated automatically, and the MSI Type Number setting is set to this number. For example, launching an executable file whose fully qualified path is stored in the **Property** table is calculated as custom action type 50. To learn what value is used for any of the basic types of custom actions, see Summary List of All Custom Action Types in the Windows Installer Help Library.

For more information about each of the types of custom actions that are available in InstallShield, see Custom Action Types.

Custom Action Return Values



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Depending on your selections in the Custom Action Wizard's Additional Options panel, Windows Installer will wait for a successful return code before continuing with the installation. Your custom action must return one of these values according to its status.

Table 4-5 • Custom Action Return Values

Return Value	Description
ERROR_FUNCTION_NOT_CALLED	The custom action was not executed.
ERROR_INSTALL_FAILURE	The custom action failed.
ERROR_INSTALL_SUSPEND	The custom action was suspended, but it will be resumed later.
ERROR_INSTALL_USEREXIT	The custom action could not complete successfully because the end user aborted it.
ERROR_NO_MORE_ITEMS	The custom action succeeded, but the remaining actions should be skipped.
ERROR_SUCCESS	The custom action succeeded.

If you are authoring an InstallScript custom action, your entry-point function can return either ERROR_SUCCESS or ERROR_INSTALL_FAILURE. These constants are defined for you when you include **Iswi.h** in your script. The InstallScript engine reports any of the other exceptions to Windows Installer.

In-Script Execution



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you create a custom action, one setting that must be configured is the In-Script Execution setting on the Respond Options panel of the Custom Action Wizard. Actions are executed in the order in which they appear in a sequence. However, the sequences are run many times during a typical installation. The In-Script Execution setting enables you to select which iteration of the sequence will trigger your action.

(Terminal Server Aware)

If you select a Terminal Server Aware option—for example, Immediate Execution (Terminal Server Aware)—the custom action impersonates the end user during per-machine installations on terminal server machines.

Immediate Execution

As its name implies, Immediate Execution runs when the internal Windows Installer installation script is being compiled. When an .msi file is launched, the Windows Installer service converts all the tables in the installation database into an internal script. (This internal script is not the same as InstallScript code.) This script is built by cycling through all the actions in the installation in the order in which they appear. The building of this script is immediate execution. When an action is encountered whose In-Script Execution setting is set to Immediate Execution, that action is executed. Therefore, this action launches before any file transfers are encountered, possibly even before the end-user interface for the installation is fully loaded.

As a rule, custom actions scheduled for immediate execution do not modify the target system, but only set properties and query the target system (for example, to see if the target system meets your product's system requirements). Custom actions that set Windows Installer properties must be scheduled for immediate execution, and custom actions that occur in the User Interface sequence must be scheduled for immediate execution.

Because actions of this type run before any changes have been made to the system, they cannot rely on files that are being installed by the installation.

Deferred Execution

Deferred Execution takes place when the internal script generated during Immediate Execution is executed by the Windows Installer service. After that script has been fully generated, the Windows Installer service runs the newly compiled script. The script runs through all of the actions in your sequences and executes them in order. However, if an action is scheduled for immediate execution, the action does not run again during Deferred Execution.

Actions that launch during Deferred Execution have access to files being installed as part of the system. As a result, you can call a custom action that calls a function from a DLL file installed with your product during this phase of the installation. However, Deferred Execution custom actions must take place between InstallInitialize and InstallFinalize in order to work properly.

Custom actions that rely on a file being installed to the target system, or that rely on other system changes to have already been performed, must be scheduled for deferred execution.

Rollback Execution

Rollback occurs when an error is encountered by the installation or the end user cancels the installation before it has a chance to complete. The Rollback Execution option allows you to set your action to execute only during rollback. Therefore, any actions that are enabled for Rollback Execution are written to the installation script, as are Deferred Execution actions. Unlike Deferred Execution actions, Rollback Execution actions launch only during rollback. (Rollback custom actions run only if the installation fails during deferred execution.)

Any custom actions that make changes to the target system during an installation should be undone with a Rollback Execution custom action in the event of rollback. For example, if you have a custom action that creates a file, you should create a second custom action that deletes the file, and schedule the second action for rollback execution. (A rollback custom action should be scheduled before the custom action it reverses.)

Commit Execution

Commit Execution actions do not run until the InstallFinalize action completes successfully. This means that they wait until the installation has completed transferring files, registering COM servers, and creating shortcuts and registry entries. Then, any actions that are set to Commit Execution launch in the order in which they appear in the sequence.

For example, if you have a custom action that creates a temporary file, you should create a second custom action that deletes the file, and schedule it for commit execution.

Deferred Execution in System Context

Like Deferred Execution actions, these actions do not launch until the script generated by the Windows Installer service is run. However, actions of this type execute with no user impersonation.

Documenting the Behavior of Custom Actions



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



Windows Logo • *The intended behavior of each custom action must be documented for the Windows logo program. This is especially helpful if system administrators deploy your product to enterprise environments; they sometimes need to know what the custom actions do. If you validate your installation package but you have not specified a value for the Help File Path setting, InstallShield generates validation error ISICE10. For more information, see ISICE10.*

Task ***To document the behavior of a custom action in your project:***

1. Create a file that describes the intended behavior of the custom action. The file should be a text-based file such as a .txt, .htm, or .rtf file. Note that each custom action should have its own document.
2. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in DIM, Merge Module, and MSM Database projects).
3. In the **Custom Actions** explorer, click the action that you are documenting.
4. For the **Help File Path** setting, click the ellipsis button (**...**) to browse to the file that describes the behavior of the custom action.

Tip *• You can specify whether InstallShield should stream the contents of each of the custom action help files into the .msi file at build time. For more information, see the description of the Include Custom Action Help setting for a product configuration in the Releases view.*

Guidelines for Creating Custom Actions that Meet Requirements of the Windows Logo Program

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you are applying for the Windows logo, your application must meet Microsoft's Windows logo program requirements. You can use the validation suite to verify whether your installation package meets most of the custom action–related logo requirements. However, some of the requirements cannot be verified through the validation suites.

Following is a list of the requirements that are not validated through the validation suites or the associated ISICEs but still must be met to achieve logo compliance for the Windows logo program:

- Do not call the Global Assembly Cache tool (**Gacutil.exe**) from a custom action. This tool was not designed to be used during installations.
- All custom actions must record success or failure in the Windows Installer log by calling **MsiProcessMessage**.
- Custom actions that change system state during installation must remove or restore the system state during uninstallation. In addition, custom actions that change system state must be written as a deferred and rollback custom action pair.

This does not apply to custom action types 19 (displays an error, returns failure, and ends the installation), 35 (sets the installation directory), or 51 (sets a property)

- The intended behavior of each custom action in your installation must be documented. This does not apply to custom action types 19 (displays an error, returns failure, and ends the installation), 35 (sets the installation directory), or 51 (sets a property). For more information, see Documenting the Behavior of Custom Actions.

  For information about all of the InstallShield custom actions that are added automatically to InstallShield projects to support different functionality, see InstallShield Custom Action Reference.

Conditionally Launching Custom Actions Based on Release Flags



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

When your installation is built, InstallShield automatically adds a property called **ISReleaseFlags** to the **Property** table. This property contains all of the release flags that are included in the build. These flags appear exactly as they do in the Release Flags setting in the Releases view. Multiple flags are separated by a comma.

A custom action can be conditionally launched based on release flags in the following ways.

Configuring the Condition Setting

The easiest way to conditionally launch a custom action is to set the Condition setting when you insert the custom action into a sequence. In this case, you can enter a condition such as **ISReleaseFlags><"MyReleaseFlag"** in the Condition setting. This condition succeeds if the substring **"MyReleaseFlag"** is contained in the value of the **ISReleaseFlags** property.

Calling MsiGetProperty

A more robust way to set a condition on your custom action based on release flags is to author a call to **MsiGetProperty** within the custom action. In addition to determining if the action should launch, you can also indicate what code is executed by the action. For example, you can use one function from a DLL file for all scenarios, yet provide different functionality for each scenario through conditional logic within your function.

The drawback to conditionally launching your custom action in this way is that you can use only a script or DLL custom action. Additionally, your custom action still launches behind the scenes. When the custom action launches, it can check if a specified release flag has been included in the build. If that flag exists, the custom action continues. If the flag is not there, the action exits.

Placing Files in the .msi Database and Extracting Them During Run Time



Project *• This information applies to the following project types:*

- *Basic MSI*

- *InstallScript MSI*

The Support Files view (in Basic MSI projects) and the Support Files/Billboards view (in InstallScript MSI projects) enable you to store temporary files that are to be used by your installation program but are not to be installed. Examples are license text files (as displayed by **SdLicense**) or DLL files called by your installation. The location to which the support files are extracted at run time is stored in the Windows Installer property **SUPPORTDIR**.



***Note** • Deferred, commit, and rollback custom actions in Basic MSI and InstallScript MSI installations have access to only some of the built-in Windows Installer properties: **CustomActionData**, **ProductCode**, and **UserSID**. If you want a custom action to access any other properties (such as **SUPPORTDIR**) during deferred, commit, or rollback execution, you need to pass them as **CustomActionData**. To learn more, see Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions.*



***Project** • Note that the value of the Windows Installer property **SUPPORTDIR** is not the same as the value of the InstallScript system variable SUPPORTDIR.*

Custom Action Script Examples

InstallScript

To access a particular support file during installation, Basic MSI projects and InstallScript MSI projects should reference the Windows Installer property **SUPPORTDIR**. You can use **MsiGetProperty** to query the path, and then append the file name to the **SUPPORTDIR** value to obtain the complete path of the file. In the settings for a custom action, this is just [SUPPORTDIR].

In an InstallScript custom action, you could use code such as the following:

```
export prototype STRING GetSupportFilePathMSI(HWND);

function STRING GetSupportFilePathMSI(hMSI)
    STRING szSupportDir[MAX_PATH + 1];
    STRING szMyFile[MAX_PATH + 1];
    NUMBER nLength;
begin
    // set initial buffer size
    nLength = MAX_PATH + 1;
    MsiGetProperty(hMSI, "SUPPORTDIR", szSupportDir, nLength);

    // reset buffer-size variable
    nLength = MAX_PATH + 1;
    MsiGetProperty(hMSI, "MYFILE", szMyFile, nLength);

    // return full file path
    return szSupportDir ^ szMyFile;
end;
```

In this example, the name of the support file that is being used is stored in the MYFILE property.

C++

In C++, you could use the following code:

```
UINT __stdcall ShowSupportdir(MSIHANDLE hInstall)
{
    TCHAR szSupportDir[MAX_PATH + 1] = {'\0'};
    DWORD dwBuff = sizeof(szSupportDir);
    MsiGetProperty(hInstall,"SUPPORTDIR",szSupportDir,&dwBuff);
    MessageBox(NULL,szSupportDir,"SUPPORTDIR is ...",MB_OK);
}
```

VBScript

In VBScript, you could use the following code:

```
dim strSupportDir
strSupportDir = Session.Property("SUPPORTDIR")
```

Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Deferred, commit, and rollback custom actions in Basic MSI and InstallScript MSI installations have access to only some of the built-in Windows Installer properties: **CustomActionData**, **ProductCode**, and **UserSID**. If you want a custom action to access any other properties during deferred, commit, or rollback execution, you can pass them as **CustomActionData**. You can do so by scheduling an immediate set-a-property type of custom action to set a property that matches the name of the custom action. The value of this property is then available in the **CustomActionData** property within the deferred, commit, or rollback custom action.

Using CustomActionData to Access a Property

The following example shows how to access the Windows Installer property **SUPPORTDIR** through a deferred InstallScript custom action.



Task ***To access SUPPORTDIR through a deferred InstallScript custom action:***

1. In the **Custom Actions and Sequences** view, create a set-a-property custom action (type 51) called **GetSUPPORTDIR**. Configure the **Property Name**, **Property Value**, and **Install Exec Sequence** settings for the custom action as follows, and leave all of the other settings blank.
   - Property Name: DisplaySupportDir
   - Property Value: [SUPPORTDIR]
   - Install Exec Sequence: After InstallInitialize
2. In the **InstallScript** view, create a new function called **DisplaySupportDir**.
3. Add the following code to display a message box containing the value of **SUPPORTDIR**:

   ```
   function DisplaySupportDir(hMSI)
   ```

```
   STRING supportDirPath;
   NUMBER supportDirPathBuffer;
begin
   supportDirPathBuffer = MAX_PATH;
   if(MsiGetProperty(hMSI, "CustomActionData", supportDirPath, supportDirPathBuffer) ==
ERROR_SUCCESS) then
     SprintfBox(INFORMATION,"Deferred Execution","The value of SUPPORTDIR is %s",supportDirPath);
     SprintfBox(INFORMATION,"Deferred Execution","The value of InstallScript's SUPPORTDIR is
%s",SUPPORTDIR);
   endif;
end;
```

4. In the **Custom Actions and Sequences** view, create an InstallScript custom action called **DisplaySupportDir**. Configure the **Function Name**, **In-Script Execution**, and **Install Exec Sequence** settings for the custom action as follows, and leave all of the other settings blank.

   - Function Name: DisplaySupportDir
   - In-Script Execution: Deferred Execution in System Context
   - Install Exec Sequence: After GetSUPPORTDIR

Important • *The property name that you enter in step 1 must match the name of the custom action that you create in step 4.*

Using CustomActionData to Access More Than One Property

If you want a deferred, commit, or rollback custom action to access more than one Windows Installer property, you can "pack" the properties in **CustomActionData** and then have your deferred, commit, or rollback custom action "unpack" them after retrieving the value of **CustomActionData**.

For example, if you want to retrieve the values of [INSTALLDIR], [SUPPORTDIR], and [SetupType], you would set the Property Value setting of your type 51 custom action to this:

[INSTALLDIR];[SUPPORTDIR];[SetupType]

where each property is separated by a semicolon.

You can use code such as the following VBScript to "unpack" the values from the **CustomActionData** property:

```
Dim PropArray
PropArray = Split(Session.Property("CustomActionData"), ";")
INSTALLDIR = PropArray(0)
SUPPORTDIR = PropArray(1)
SetupType = PropArray(2)
'Now do something with these variables...
```

InstallScript Custom Actions

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *Merge Module*

In order to use your InstallScript functions in a Windows Installer installation, you first need to create a custom action that calls your script. The easiest way to create a custom action is to use the Custom Action Wizard.

The primary panels of this wizard are:

- Action Type Panel
- Action Parameters Panel
- Respond Options Panel
- Insert into Sequence Panel

When you complete the wizard, a custom action of type 65536 is created. This custom action type is not defined in the Windows Installer Help Library. Because InstallScript is not natively supported by Windows Installer, InstallShield created action type 65536.



***Project** • You should not use an InstallScript custom action in the User Interface sequence of an InstallScript MSI project. You can use the event-driven script to provide the user interface in an InstallScript MSI project.*

Calling Functions in Standard DLL Files



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Although Windows Installer restricts the parameters that you can pass to a functions in a DLL file written for a custom action, InstallShield offers a solution that enables you to pass any number of parameters to a function and store the return value. Note that the function must use the __stdcall calling convention. Functions with more than one parameter will not work properly if this convention is not used.

You can specify that you are calling a function in a standard DLL file in the Custom Action Wizard's Action Type panel by selecting **Call a function in a standard dynamic-link library** for the Type option. The next panel, the Function Definition panel, allows you to specify the function's parameters and return value.



***Note** • If you call a function in a standard DLL file that is installed with the product and the action is scheduled as deferred (for the In-Script Execution setting), the DLL file that you are calling must be the component's key file.*

*A custom action that calls a function in a standard DLL file stored in the **Binary** table cannot be scheduled for deferred execution.*

If you want a Windows Installer property to be set to the function's return value, the action must be configured for immediate execution. If you try to specify a return property for an action that is scheduled for deferred, rollback, or commit execution, the return property is ignored at run time.

Understanding the Parameters and Return Value as a Formatted String

When you specify the function signature for a standard DLL file action, that action's Function Signature setting in the Custom Actions and Sequences view uses the following formatting to display the function's name, arguments, return type, and return property.

```
DataType1=[PropertyName1] DllName::FuncName(in DataType2="Value", Direction DataType3=[PropertyName2])
```

The following table describes each portion of the format.

Table 4-6 • Format for the Arguments of a Function

Argument	Description
DataType1=[PropertyName1]	`DataType1` is of type void, STRING, BOOL, NUMBER, HWND, HANDLE, or POINTER. `PropertyName1` is the name of a property from the Property Manager view. If the return type is void, `=[PropertyName1]` is omitted.
DllName	Specify the name of the DLL file, without the dot or file extension.
FuncName	Enter the name of the function that you are calling.
in DataType2="Value"	This is the format for an argument that is a constant. `DataType1` is of type STRING, BOOL, NUMBER, HWND, HANDLE, or POINTER. `Value` is the data that you want to pass for this argument. The constant's data type is always preceded by `in`.
Direction DataType3=[PropertyName2]	This is the format for an argument that references a Windows Installer property's value. `Direction` can be *`in`*, *`inout`*, or *`out`*. For a discussion of these property types, see the Function Definition panel under the argument's source. `DataType3` can be of type STRING, BOOL, NUMBER, HWND, HANDLE, or POINTER. `PropertyName2` is the name of a property whose value will be passed to or set by the function, depending on the value of `Direction`.

Example

For the custom action created in Calling MessageBoxA in an Installation, the Target value reads:

```
void User32::MessageBoxA(in HWND=0, in STRING=[MESSAGEPROP], in STRING=[CAPTIONPROP], in NUMBER=1)
```

In this example, the data type is *`void`*, *`in HWND=0`* represents a numeric value of 0, and *`MESSAGEPROP`* is a property that contains a string that will be passed to the function MessageBoxA in **User32.dll**.

Calling Functions in Windows Installer DLL Files



Project • *This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

A DLL file function specifically written for a Windows Installer custom action can accept only the handle to the installer database as a parameter. The steps below explain how you can retrieve information from the .msi database for use in your custom action.

You specify that your custom action resides in a Windows Installer DLL file (custom action types 1 and 17) by selecting **Call a function in a Windows Installer dynamic-link library** in the Custom Action Wizard's Action Type panel.

An alternative is to select **Call a function in a standard dynamic link library** in the Action Type panel. In this case, InstallShield allows you to specify the function's parameters.



Task ***There are three major steps involved in passing a parameter to a function in a DLL file written for Windows Installer:***

1. Prepare your DLL file.
2. Create a custom action and insert it into one of the sequences.
3. Pass the parameter using the Property Manager view. These steps are explained in greater detail below.

Preparing DLL Files

In order for you to pass data to a DLL file function in a custom action, the function to which you are passing data needs to call the **MsiGetProperty** function, which retrieves the value of an installer property. In the example below, it retrieves the value of a public property called **MYPROPERTY**.

```
UINT __stdcall MyActionName(MSIHANDLE hInstall)
{
    TCHAR szValue[51] = {0};
    DWORD dwBuffer = 50;

    MsiGetProperty(hInstall, TEXT("MYPROPERTY"), szValue, &dwBuffer);

    MessageBox(GetForegroundWindow( ),
        szValue,
        TEXT("Value of MYPROPERTY"), MB_OK | MB_ICONINFORMATION);

    return ERROR_SUCCESS;
}
```

For more information, see MsiGetProperty in the Windows Installer Help Library.

If you use a C++ compiler to build your DLL file, ensure that the compiler does not change the function names exported from the DLL file. With Microsoft Visual C++, for example, you can create a .def file that specifies the function names to be exported from your DLL file. A typical .def (called MyActions.def) file looks like the following:

```
LIBRARY MyActions
EXPORTS
```

```
MyActionName
```

For more information about function name decoration, see your compiler documentation.

Creating Custom Actions and Inserting them into Sequences

The second step is to create your custom action and insert it into one of the Installation sequences. To create the custom action, use the Custom Action Wizard, and to place the custom action, use the Custom Actions and Sequences view.

Passing the Parameter Using the Property Manager



Task ***To pass the new parameter using the Property Manager:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Click the **New Property** button. InstallShield adds a new row at the bottom of the view.
3. In the **Name** column, type the name of the property that you would like to retrieve with **MsiGetProperty** (**MYPROPERTY**, in this example).
4. In the **Value** column, type the value that you want to pass. For example, if you want to pass the URL to your Web site, type **http://www.mycompany.com**.

Passing Parameters to a DLL File Function in a Custom Action



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

There are a number of reasons why you might pass a parameter to a function in a custom action. For example, you might want to pass a URL for Web registration or the UserName property for a custom user interface. However, Windows Installer does not directly support passing parameters to DLL file functions.

The entry-point function for a Windows Installer DLL file can have only one argument, which is the handle to the database. To learn about a suggestion for passing parameters to a DLL file written for Windows Installer, see Calling Functions in Windows Installer DLL Files.



Task ***To accomplish this in InstallShield:***

1. In the Custom Wizard's **Action Type** panel, select **Call a function in a standard dynamic-link library**.
2. In the **Function Definition** panel, specify the parameters that you want InstallShield to pass to the function.

Calling a Public Method in a Managed Assembly

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The managed-code type of custom action calls a public method in a .NET assembly that is written in managed code such as Visual Basic .NET or C#.

If you include a managed-code custom action in your project, InstallShield creates a C++ Windows Installer wrapper DLL for your .NET assembly. The wrapper DLL includes your assembly, as well as the information that is required to mediate, load, and run the assembly.

Note *• If you specify that the location of your managed assembly should be the* ***Binary*** *table, InstallShield stores the wrapper DLL—which includes your assembly—in the* ***Binary*** *table.*

Run-Time Requirements for Managed-Code Custom Actions

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The managed-code type of custom action requires the Microsoft .NET Framework on the target system.

InstallShield includes redistributables for the .NET Framework. If it is possible that target systems may not have the .NET Framework, you can add the appropriate .NET Framework redistributable to your project. For instructions, see Adding .NET Framework Redistributables to Projects.

You can use the property **IS_CLR_VERSION** to identify a semicolon-delimited list of .NET Framework versions that the custom action should attempt to load to run your managed code. In most scenarios, this property is not set in the installation package. It is set at the command line. To specify that version 1.1 is required, use the following command-line parameter:

```
IS_CLR_VERSION=v1.1.4322
```

Note that the complete version number of the .NET Framework should be specified for the property value. If more than one version is acceptable, you can specify a semicolon-delimited list of versions. The first one that can be loaded is used. For the following example command-line parameter, the custom action attempts to load version 2.0. If that version is not present, the custom action attempts to load version 1.1. If version 1.1 is not present, the custom action fails.

```
IS_CLR_VERSION=v2.0.50727;v1.1.4322
```

To specify that the custom action should attempt to load whatever is the latest version of the .NET Framework that is installed if none of the specified versions are installed, add a semicolon to the end of the property value, as shown in the following example:

```
IS_CLR_VERSION=v2.0.50727;v1.1.4322;
```

The semicolon at the end of the property value also indicates that if none of the specified versions are present but a version of the .NET Framework is already loaded, the custom action uses the currently loaded version, even if it is not the latest version that is installed.



Note *• If the execution of your managed-code custom action is set to deferred mode, you must use the Windows Installer property* ***CustomActionData*** *to pass the* ***IS_CLR_VERSION*** *property and value. To learn more, see Specifying the Signature for a Managed Method in an Assembly Custom Action.*



Tip *• If issues with the custom action occur at run time because of .NET Framework version mismatches, you may want to instruct end users to set the* ***IS_CLR_VERSION*** *property at the command line when they run your installation. Note that for deferred custom actions, you must have already used the* ***CustomActionData*** *property to pass the* ***IS_CLR_VERSION*** *property. For more information, see Specifying the Signature for a Managed Method in an Assembly Custom Action.*

Specifying the Signature for a Managed Method in an Assembly Custom Action



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The Managed Method Definition panel in the Custom Action Wizard is where you specify arguments for the method that your managed assembly custom action should call. You can also specify this information on the Method Signature dialog box.

Using the Default Method Signature

If you use the default method signature, the custom action handles methods with up to one parameter. If the method takes a parameter, it is passed the MsiHandle, as well as any return type. If the method returns an integer, this integer is passed directly to Windows Installer. Therefore, if a method returns integer 1603 (ERROR_INSTALL_FAILURE) in this scenario, the installation aborts.

For more information, see Custom Action Return Values in the Windows Installer Help Library.

Using a Custom Method Signature for an Immediate Custom Action

If you use a custom method signature, the custom action calls the method with the arguments that you specified. If the value of a parameter is a Windows Installer property and the custom action is set for immediate execution, the method stores any by-reference and out parameter values in the passed properties. In addition, the custom action converts the return value to a string and stores it in the return property. If no return property is specified, the return value is ignored.

Note that if you use a custom signature, the custom action's return value is not passed to Windows Installer. However, if the managed code throws an unhandled exception, the custom action returns ERROR_INSTALL_FAILURE to Windows Installer. Therefore, in order for the custom action to indicate failure and for Windows Installer abort the installation, the managed code must throw an unhandled exception.

Using a Custom Method Signature for a Deferred, Commit, or Rollback Custom Action

Deferred, commit, and rollback custom actions have access to only some of the built-in Windows Installer properties: **CustomActionData**, **ProductCode**, and **UserSID**. Therefore, if you use a custom method signature and you want your managed assembly custom action to access or pass any other properties during deferred, commit, or rollback execution, you must pass them through the **CustomActionData** property. Note the following guidelines for deferred, commit, and rollback managed assembly custom actions:

- Custom signatures that pass properties need to have their values passed in as **CustomActionData** in the following format:

  PROPERTYNAME="Property Value"

  Separate multiple property name–property value pairs with a space, as in the following example:

  PROPERTYNAME1="Property Value 1" PROPERTYNAME2="Property Value 2"

- If the managed assembly for your deferred, commit, or rollback custom action is installed with the product, use **CustomActionData** to identify the location of the managed assembly, and use the following format:

  #filekey.dll="location of assembly"

  For the "location of assembly" portion, use the format **[#filekey.dll]**.

- If the path for the managed assembly references one or more properties, use **CustomActionData** to pass each of the properties and their corresponding values.

- Consider using **CustomActionData** to pass the following property and value:

  IS_CLR_VERSION="[IS_CLR_VERSION]"

  If run-time issues occur because your custom action attempts to load the wrong version of the .NET Framework to run your managed code, you can have end users set the value of the **IS_CLR_VERSION** property at the command line when they run your installation. For deferred, commit, and rollback managed-code custom actions, **IS_CLR_VERSION** must be passed to **CustomActionData** in order for end users to use this run-time override. To learn more about the **IS_CLR_VERSION** property, see Run-Time Requirements for Managed-Code Custom Actions.

Specifying Parameter Values for a Custom Method Signature

If you use a custom method signature, you can enter any of the following types of values for a parameter:

- A single property that is enclosed within square brackets—for example, **[ProductName]**. Note that immediate-mode managed assembly custom actions can update ref and out parameters. InstallShield does not support combining multiple properties or mixing properties with strings for a parameter value.
- A literal—for example, a string such as the number **1** or **1603**.
- An explicit string that is enclosed within quotation marks—for example, **"My Project Name-1"**.
- The variable *MsiHandle*, which indicates the installation handle.

At run time, after a property has been resolved or a string has been isolated from within its quotation marks, the custom action attempts to convert the string to an instance of the type of the method's appropriate parameter. A parameter of type String is passed as is, and all normal numeric types are handled by calling the public static Parse(string) method to turn the string into the type. In addition, the custom action calls any type's public static Parse(string) method and uses the return value as the passed parameter value. IntPtr is also handled despite the lack of a Parse method.

Using 32-Bit vs. 64-Bit Managed-Code Custom Actions



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

When you build a release that includes a managed-code custom action in your project, InstallShield attempts to determine the target architecture (32 bit or 64 bit) of the main .NET assembly that is associated with the custom action. InstallShield does this by examining the portable executable (PE) file for the assembly and determining if the PE file is 32 bit or 64 bit. If the PE file uses Microsoft intermediate language (MSIL) code, InstallShield treats it as 32 bit. InstallShield configures the release so that the appropriate version of the .NET Framework—32 bit or 64 bit—is used to run your managed code at run time.

If you want to override the built-in default behavior, use the Direct Editor view to add a new record with the following fields to the **ISClrWrap** table.

Table 4-7 • Overriding the Default Architecture for a Managed-Code Custom Action

ISClrWrap Table Column	Description
Action_	Enter the name of the managed-code custom action that you want to modify.
Name	Enter the following: **TargetPlatform**
Value	Specify the appropriate architecture. • To use a 32-bit version of the .NET Framework, enter the following: **x86** • To use a 64-bit version of the .NET Framework, enter the following: **x64**

Calling a Kill-Process Custom Action

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The kill-process type of custom action terminates one or more processes at run time. The following procedure describes how to include and configure this type of custom action in your project.

Task ***To add a kill-process custom action to your project:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Right-click the **Custom Actions** explorer and then click **New Kill Process**. InstallShield adds a kill-process custom action with a default name.
3. Enter a new name, or right-click it later and click **Rename** to assign it a new name. Use a name that helps you distinguish this custom action from other actions in your project.
4. In the right-pane, configure the action's settings:
   a. In the **In-Script Execution** setting, select the iteration of the sequence that should trigger the action. For details about each option, see In-Script Execution.
   b. Use the settings in the **Sequences** area to schedule the custom action at the point where you want the process to be terminated. As an alternative, you can drag the custom action from the **Custom Actions** explorer to the appropriate node under the Sequences explorer.
   c. In the **Function Name** setting, select the appropriate function:
      - **KillProcess**—If you selected one of the immediate options in the **In-Script Execution** setting and you want to kill a process that has a particular name, select this option.
      - **KillProcessByID**—If you selected one of the immediate options in the **In-Script Execution** setting and you want to kill a process that has a particular process identifier (PID), select this option.
      - **KillProcessDeferred**—If you selected one of the deferred, commit, or rollback options in the **In-Script Execution** setting and you want to kill a process that has a particular name, select this option.
      - **KillProcessByIDDeferred**—If you selected one of the deferred, commit, or rollback options in the **In-Script Execution** setting and you want to kill a process that has a particular PID, select this option.
   d. In the **Processes** setting, enter executable file names or PIDs of the processes that you want to terminate.

If Function Name is set to:	Then, specify Processes using:
KillProcess or KillProcessDeferred	Executable file names
KillProcessByID or KillProcessByIDDeferred	PIDs

If you want to terminate more than one process, use a semicolon (;) to separate each file name or PID. For example, to terminate processes that have names such as **myfile1.exe** and **myfile2.exe**, enter the following:

myfile1.exe;myfile2.exe

Tip *• The value of the* ***Processes*** *setting may by written to the* ***ISTerminateProcesses*** *property. If you have additional kill-process custom actions that do not specify a value in the* ***Processes*** *setting, such as those migrated from InstallShield 2015 or earlier, shared use of the* ***ISTerminateProcesses*** *property may result in undesired behavior.*

e. Configure any of the action's other settings as needed.

Calling a PowerShell Custom Action

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Windows PowerShell is a .NET Framework–based command-line shell and script language that enables system administrators to automate system configuration tasks. InstallShield lets you include in your installations custom actions that run PowerShell scripts (.ps1). You may want to add this type of custom action to a project to perform system configuration tasks at installation run time. Note that PowerShell actions require PowerShell 2.0 or later on target systems.

Adding a PowerShell Custom Action to Your Project

Task ***To add a PowerShell custom action to your project:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Right-click the **Custom Actions** explorer, point to **New PowerShell**, and click one of the following commands:
   - **Stored in Binary table**—To have your code base stored in the **Binary** table, select this command. This location is useful if you do not want the file to be installed on the target system.
   - **Installed with product**—To call code from a script file that is going to be installed on the target system, select this command.

   InstallShield adds a PowerShell custom action with a default name.
3. Enter a new name, or right-click it later and click **Rename** to assign it a new name. Use a name that helps you distinguish this custom action from other actions in your project.
4. In the **PowerShell Script File Name** setting, select the PowerShell script file (.ps1) in the list of files that are stored in the **Binary** table or that are included in your project. If the location that you specified is stored in the **Binary** table, you can click this ellipsis button (...) in this setting to browse to the file.
5. Configure the action's other settings as needed.

Once you have added the custom action to your project, schedule it as needed. For more information, see Inserting Actions into Sequences.

Run-Time Requirements for a PowerShell Custom Action

PowerShell is installed on only some operating systems by default. If an installation launches a PowerShell custom action but PowerShell is not installed, the custom action fails.

To check that PowerShell is installed on a target system, you can add the predefined system search for PowerShell to your project, and configure your PowerShell custom action to run only if the system search determines that PowerShell is installed. You can also use this system search to determine which version of PowerShell is installed, and you can include a condition for the custom action that triggers its launching only on appropriate target systems. By default, the predefined system search for PowerShell sets the value of the **POWERSHELLVERSION** property to the version of PowerShell.

You can use the Windows Installer property **IS_CLR_VERSION** to identify a semicolon-delimited list of .NET Framework versions that the custom action should attempt to load to run your PowerShell script.

To learn more about the **POWERSHELLVERSION** and **IS_CLR_VERSION** properties, see Windows Installer Property Reference.

Interacting with the Running Basic MSI or InstallScript MSI Installation

Several cmdlets that let you interact with the running Basic MSI or InstallScript MSI installation are available:

Table 4-8 • Cmdlets that Interact with a Running Basic MSI or InstallScript MSI Installation

Cmdlet	Description
get-property -name [Property Name]	This cmdlet gets the value of a Windows Installer property. The Name parameter is a positional parameter. The following sample code gets the value of the property called **MYPROPERTY**: `$Value = get-property -name MYPROPERTY`

set-property -name [Property Name] -value [value]	This cmdlet sets the value of a Windows Installer property. The Name and Value parameters are positional parameters.
	
	The following sample code uses the value of the **MYPROPERTY** property to set the value of a second property called **NEWPROPERTY**:
	
	`$Value = get-property -name MYPROPERTY`
	`set-property -name NEWPROPERTY -value $Value`

format-properties -Value [format string]	This cmdlet expands the value of a formatted expression. The Value parameter is a positional parameter. The following sample code formats a string that contains a Windows Installer property called **NEWPROPERTY**. The property name and its surrounding square brackets are replaced by the value of the property at run time. `$myFormat = format-properties -Value "This is a text string with [NEWPROPERTY] embedded."`

Table 4-8 • Cmdlets that Interact with a Running Basic MSI or InstallScript MSI Installation (cont.)

Cmdlet	Description
trace-info -LogMessage [log string]	This cmdlet writes any information that is needed to the Windows Installer log, making it available for debugging or other information purposes. The LogMessage parameter is a positional parameter. The following sample code writes the string, **I am a PowerShell custom action** to the Windows Installer log. `trace-info -LogMessage "I am a PowerShell custom action"`

To indicate that your script has succeeded, ensure that it returns 0. For example, you may want to end your script with the following:

```
exit(0)
```

Launching Executable Files



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Launching an executable file from your installation is useful to install third-party tools, display a readme file, or display a Web page that contains the most up-to-date information about the product being installed. To launch an executable file from within your installation, you need to add a custom action in the Custom Actions and Sequences view (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or the Custom Actions view (in DIM, Merge Module, and MSM Database projects).

You can use the Custom Action Wizard to create the custom action that launches an executable file from your installation. Each wizard panel is listed below, with the entries that you need to make in order to launch an executable file.

Basic Information Panel

Table 4-9 • Settings for the Basic Information Panel in the Custom Action Wizard

Panel Option	Value
Name	Provide a meaningful name for your custom action. This name is used internally in your product and is for identification purposes only.
Comment	Type comments about this custom action.

Action Type

Table 4-10 • Settings for the Action Type Panel in the Custom Action Wizard

Panel Option	Value
Type	Select the type of custom action that you want to create. For this example, select Launch an executable.
Location	The selection that you make depends on where the target executable file will be during the installation. Because Notepad is on everyone's machine, it is the target for this custom action. Because Notepad is practically guaranteed to be present in the target machine's Windows folder, you can point to it using the **Directory** table. Therefore, select **Stored in the Directory** table.

Action Parameters

Table 4-11 • Settings for the Action Parameters Panel in the Custom Action Wizard

Panel Option	Value
Source	Since you choose to launch an executable file stored in the **Directory** table, you are given a list of choices that reflect all of your current entries in the **Directory** table. One of these options is **WindowsFolder**. Choose this option to point to Notepad without having to hard-code a path.
Target	For this option, you need to enter the target file within the specified directory from the Source option. Enter **Notepad.exe** and click the Next button to continue.

Additional Options

Table 4-12 • Settings for the Additional Options Panel in the Custom Action Wizard

Panel Option	Value
Return Processing	Specify how Windows Installer should control the processing of the custom action thread. You can have the main and custom action threads run synchronously (the installation waits for the custom action thread to complete before resuming the main installation thread) or asynchronously (the installation runs the custom action simultaneously as the main installation continues). In this example, select the **Synchronous (Check exit code)** option.

Respond Options

Table 4-13 • Settings for the Respond Options Panel in the Custom Action Wizard

Panel Option	Value
In-Script Execution	Select Immediate execution to have your action executed as soon as it is encountered in the script.
Execution Scheduling	Select **Always execute** so that your custom action launches every time that it is encountered.

Inserting Custom Actions into Sequences

After you have created a custom action, you must insert it into a sequence in the Custom Actions and Sequences view.

Using Msiexec.exe to Launch a Second Windows Installer Installation



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

An alternative to launching a second .msi package using the nested installation custom action type is to create a custom action that launches **Msiexec.exe** instead. Launching a second installation in this manner causes it to run in its own process and creates the proper entries on the target system. In addition, the uninstallation process is much more effective when you launch your second installation in this way. Registry entries are properly cleaned up and reference counts are incremented and decremented accurately.

Creating a Custom Action

The first step is to create your custom action. The easiest way to do this is to use the Custom Action Wizard.



Task ***To create a custom action using the wizard:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in DIM, Merge Module, and MSM Database projects).
2. Right-click the **Custom Actions** explorer and click **Custom Action Wizard**. The **Custom Action Wizard** opens.
3. On the **Basic Information** panel, specify a name and comment for your custom action and click **Next**.
4. On the **Action Type** panel, in the **Type** list, select **Launch an executable**. In the **Location** list, select **Stored in the Directory table**. The **Directory** table allows you to choose from predefined folders, such as SystemFolder, which is where **Msiexec.exe** is located.
5. On the **Action Parameters** panel, browse for the executable file (**Msiexec.exe**) of that you are launching. The **Target** box enables you to specify the name of the executable file that you would like to launch, as well as any command-line parameters that you would like to pass to it. For example:

   ```
   msiexec.exe /i "[SOURCEDIR]AnotherSetup\SecondSetup.msi" /qb
   ```

   The first part of this entry, **msiexec.exe**, is the name of the executable file that you would like to launch. The next section, **/i "[SOURCEDIR]AnotherSetup\SecondSetup.msi"**, tells **Msiexec.exe** which package to run. In this case, it is pointing to a file one folder below that of the source folder of the initial installation. Finally, **/qb** tells the Windows Installer service to run the installation with minimal user interface. Only a progress bar is displayed.
6. On the **Additional Options** panel, select the default options and click **Next** until you finish the wizard.

Inserting a Custom Action into a Sequence

After you have created a custom action, you need to insert it into a sequence.



Task ***To sequence this custom action:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in Merge Module and MSM Database projects).
2. In the **Sequences** explorer, under the **Installation** sequence, expand the **User Interface** item. The actions and dialogs that are scheduled for this sequence are listed.
3. Right-click **CostFinalize** and click **Insert**. The Insert Action dialog box opens.
4. Select the new custom action.
5. Click **OK**.

Next, verify that the installation package that you want to launch is located in the folder that you specified in the Action Parameters panel, build your installation, and run it.



***Note** • A custom action that launches* **Msiexec.exe** *must be placed in the User Interface sequence, which means that the custom action will not run if the end user runs the installation silently.*

Nested Installations



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



***Important** • Nested installations is a deprecated feature of the Windows Installer. Applications installed with nested installations sometimes fail because they are difficult for end users to service correctly. Microsoft Corporation recommends that you avoid using nested installations and nested-installation custom actions to install products that are intended to be released to the public. To learn more, see Concurrent Installations in the Windows Installer Help Library.*

A nested installation is a type of custom action that installs or removes another .msi package (sometimes called the *child product*) from within a running installation (called the *parent product*).



***Note** • Before using nested-installation custom actions, you should be aware of the following restrictions:*

- *Nested installations do not display a user interface.*
- *A child product generally does not appear in Add or Remove Programs in Control Panel on the end user's system, and it is not automatically uninstalled when the parent product is uninstalled.*

Creating Nested Installation Custom Actions



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



***Important** • Nested installations is a deprecated feature of the Windows Installer. Applications installed with nested installations sometimes fail because they are difficult for end users to service correctly. Microsoft Corporation recommends that you avoid using nested installations and nested-installation custom actions to install products that*

are intended to be released to the public. To learn more, see Concurrent Installations in the Windows Installer Help Library.



Task ***To create a nested installation custom action using the wizard:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Right-click the **Custom Actions** explorer and click **Custom Action Wizard**. The **Custom Action Wizard** opens.
3. On **Basic Information** panel, specify a name and comment for your custom action and click **Next**.
4. On the **Action Type** panel, in the **Type** list, select **Launch another .msi package**. In the **Location** list, select **Included within your main setup**.
5. On the **Action Parameters** panel, browse for the location of the .msi file that you are launching. (For simplicity, assume that the child product is packaged with all files compressed into the .msi package.) The **Target** box enables you to specify command-line switches to pass to the child installation. For example, to install all of the features of the child installation with their default settings, and to use the same value for the **ALLUSERS** property used by the parent installation, type the following in the **Target** box:

   ARPSYSTEMCOMPONENT=1 ADDDEFAULT=ALL ALLUSERS=[ALLUSERS]

   You can use similar expressions to use the same value of INSTALLDIR in the child as in the parent.
6. Accept the default options on the **Additional Options** panel and click **Next** until you finish the wizard.

Inserting Nested Installation Custom Actions into Sequences



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Important • *Nested installations is a deprecated feature of the Windows Installer. Applications installed with nested installations sometimes fail because they are difficult for end users to service correctly. Microsoft Corporation recommends that you avoid using nested installations and nested-installation custom actions to install products that are intended to be released to the public. To learn more, see Concurrent Installations in the Windows Installer Help Library.*

After you have created the custom action, you need to insert it into a sequence.



Task ***To insert a nested installation custom action into a sequence:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Sequences** explorer, under the **Installation** sequence, expand the **User Interface** item. The actions and dialogs that are scheduled for this sequence are listed.

3. Right-click **CostFinalize** and click **Insert**. The Insert Action dialog box opens.
4. Select the new custom action.
5. In the **Condition** box, type **Not Installed**. This indicates that the nested installation should be performed only the first time that the parent product is installed.
6. Click **OK**.

Removing Child Products



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Important *• Nested installations is a deprecated feature of the Windows Installer. Applications installed with nested installations sometimes fail because they are difficult for end users to service correctly. Microsoft Corporation recommends that you avoid using nested installations and nested-installation custom actions to install products that are intended to be released to the public. To learn more, see Concurrent Installations in the Windows Installer Help Library.*

A child product is not automatically removed when the parent product is removed. However, you can create a second nested-installation custom action that does this.



Task ***To create the custom action:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Right-click the **Custom Actions** explorer and click **Custom Action Wizard**. The **Custom Action Wizard** opens.
3. On the **Basic Information** panel, specify a name and comment for your custom action and click **Next**.
4. On the **Action Type** panel, in the **Type** list, select **Launch another .msi package**. In the **Location** list, select **An application that is advertised or already installed**.
5. On the **Action Parameters** panel, browse for the location of the .msi file that you want to remove. In the **Target** box, leave the default properties:

   ALLUSERS=[ALLUSERS] REMOVE=ALL
6. Accept the default options on the **Additional Options** panel and click **Next** until you finish the wizard.

Task ***To sequence this custom action:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Sequences** explorer, under the **Installation** sequence, expand the **Execute** item. The actions and dialogs that are scheduled for this sequence are listed.
3. Right-click **InstallValidate** and click **Insert**. The Insert Action dialog box opens.
4. Select the new custom action.
5. In the **Condition** box, type the following:

   REMOVE="ALL"
6. Click **OK**.

This custom action will uninstall the child product when the parent product is uninstalled.

Note *• When you are creating a nested-installation custom action of type Launch another .msi package with the Location setting set to* ***Stored on the source media****, be aware of the following:*

- *The child installation must not be compressed inside a* **Setup.exe** *installation launcher.*
- *If the child installation's files are not compressed inside the child .msi database, you must manually copy the child installation's files to the* **Disk1** *folder of the parent installation's release folder.*

Calling MessageBoxA in an Installation

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To call the Windows API function MessageBoxA() in a custom action to display a message box:***

Part A: Launch the Custom Action Wizard

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Right-click the **Custom Actions** explorer and click **Custom Action Wizard**.

Part B: Start a Custom Action

1. In the **Basic Information** panel, type **CA_Example** for the custom action's name. Click **Next**.
2. In the **Action Type** panel, for the custom action's type, select **Call a function in a standard dynamic-link library**.

3. Since **MessageBoxA** is exported in **User32.dll**, which is found on every supported Windows platform, select **Destination machine search path** in the **Location** list. Click **Next** to continue.

Part C: Provide the Function Definition

MessageBoxA() has the following syntax:

```
int MessageBoxA (hWnd, lpText, lpCaption, uType);
```

Specify this information in the Function Definition panel:

1. In the **Name** box, type **MessageBoxA**.
2. Click the last row in the **Arguments** box to specify the first parameter. Edit the fields for each parameter until they mirror the following list:

   Table 4-14 • Parameters for the Arguments Box

Type	Source	Value
HWND	Constant	MsiWindowHandle
STRING	in Property	MESSAGEPROP
STRING	in Property	CAPTIONPROP
NUMBER	Constant	1

   In the **Value** list for the HWND type, select **MsiWindowHandle**. If you set the HWND type to this value, your message box will not hide behind the installation window. The properties **MESSAGEPROP** and **CAPTIONPROP** are not available in the list. When you enter these names in the **Value** list, they are added to the Property Manager.

3. In the **Return Type** list, select **void**. (Although **MessageBoxA()** does return a number, checking that value in a property is outside the scope of this example.)
4. Click **Next** to proceed to the **Action Parameters** panel.

Part D: Specify the Source for MessageBoxA

The next significant setting in the Custom Action Wizard is the Source field in the Action Parameters panel. MessageBoxA is found in **User32.dll**. Click the Browse button to locate **User32.dll** in your Windows system folder.

In the next panel of the wizard, click Next to accept the defaults, and then click Finish in the Summary panel to dismiss the wizard and add CA_Example to your project.

Part E: Initialize the Properties' Values

For the string parameters in **MessageBoxA()**, properties were provided, as described in the above procedures. The next step is to give those properties a value:

1. In the View List under **Behavior and Logic**, click **Property Manager** to view the installer properties in your project. The properties **MESSAGEPROP** and **CAPTIONPROP** were created as a result of referencing these properties in the **Custom Action Wizard**.
2. Locate **MESSAGEPROP** and click in its **Value** column to supply a value. Enter **Can you see this text?** for the value.
3. Set the CAPTIONPROP property to **Custom Action Example**.

Part F: Insert the Custom Action into a Sequence

In order to execute the custom action, you must either place it into a sequence or make it the result of a dialog's control event.

Follow the steps below to insert CA_Example into the Installation sequence of the installation project:

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Sequences** explorer, under the **Installation** sequence, expand the **User Interface** item. The actions and dialogs that are scheduled for this sequence are listed.
3. Right-click the **MigrateFeatureStates** action (the action right before the InstallWelcome dialog) and click **Insert**. The **Insert Action** dialog box opens.
4. In the list at the top of the dialog box, select **Custom Actions** (for installation projects).
5. In the list of custom actions, select **CA_Example**.
6. Click **OK**.

CA_Example is added to the Installation sequence directly after MigrateFeatureStates.

Part G: Test the Call to MessageBoxA

1. Build a release. (The Output window may show build errors if you have not added any components or files to your project.)
2. On the **Build** menu, click **Run**.

InstallShield runs your installation. After the Welcome dialog is displayed, a Custom Action Example message box opens.

Searching for Files on the Target System

System Search View

The System Search view provides the functionality for locating a particular file, folder, registry or .ini file on a target system before installing your application.

Direct Editor: Basic MSI Projects

Windows Installer uses records in the **Signature**, **AppSearch**, and "locator" tables for instructions for searching for files on the target system. The **Signature** table contains information about the file to be located, and the **AppSearch** table specifies a property to set to the full path of the located file, if found.

For example, to search for a file called **FindMe.exe**, use the Direct Editor to add the following record to the **Signature** table.

Table 4-15 • Sample Data for the Signature Table

Table Column	Sample Data
Signature	findme_sig
FileName	FindMe.exe

Note that other fields in the **Signature** table enable you to specify optional version, size, date, and language information.

Add the following record to the **AppSearch** table.

Table 4-16 • Sample Data for the AppSearch Table

Table Column	Sample Data	Explanation
Property	LOCATION_OF_FINDME	Must be a public property.
Signature_	findme_sig	Same name used in **Signature** table.

There are four locator tables in which you can specify where Windows Installer should begin searching for the file: **CompLocator**, **RegLocator**, **IniLocator**, and **DrLocator**. To search for a file in a specific directory, use the **DrLocator** table. For example, to search for **FindMe.exe** in the user's Program Files directory, we add the following record to the **DrLocator** table.

Table 4-17 • Sample Data for the DrLocator Table

Table Column	Sample Data
Signature	findme_sig
Parent	
Path	[ProgramFilesFolder]
Depth	2

After the AppSearch action runs, the public property **LOCATION_OF_FINDME** will contain the full path to **FindMe.exe** on the end user's system if it exists; if the file is not located, this property will be undefined.

InstallScript: InstallScript MSI Projects

The **FindFile** and **FindAllFiles** functions enable you to search for existing files on the target system. For example, an implementation of the **OnAppSearch** event-handler function that searches for a file called **FindMe.exe** in the user's Program Files folder might appear as follows:

```
function OnAppSearch( )
    STRING svFoundFile;
begin
    FindAllFiles(PROGRAMFILES, "FindMe.exe", svFoundFile, RESET);
    MessageBox("Found FindMe.exe at: " + svFoundFile, INFORMATION);
end;
```

Launching the Application After the Installation Is Complete

The **LaunchApplication** function launches an executable file. For example, the following code launches a copy of **Program.exe** in the end user's INSTALLDIR folder, returning execution to the script when the end user closes the program's window.

```
LaunchApplication (INSTALLDIR ^ "Program.exe", "", "", SW_NORMAL,"", LAAW_OPTION_WAIT);
```



Project *• InstallScript projects use TARGETDIR rather than INSTALLDIR.*

Exiting the Installation from Within an InstallScript Custom Action



Project *• This information applies to the following project types:*

- *Basic MSI with InstallScript custom actions*
- *InstallScript MSI*
- *Merge Module with InstallScript custom actions*

An InstallScript entry-point function that returns the value ERROR_INSTALL_FAILURE causes the installation to exit.

Changing ODBC Properties Through Script

If you want to change ODBC properties (such as DBQ and SystemDB) through script, you must configure these options to use installer properties. Then from the script you can set those installer properties at run time. For example, you would set SystemDB to **[SYSTEM_DB_DIR]\MyDatabase** in the InstallShield interface. Then from the script, you could call **MsiSetProperty** to set the value of **SYSTEM_DB_DIR**.



Tip *• Set the property value before the OnMoving event, preferably in OnFirstUIBefore. After that, it is too late because the installer has already built up its internal script information, which cannot be modified.*

- *Use all uppercase letters for your property name.*

- *If **SYSTEM_DB_DIR** exists in the **Directory** table, omit the backslash (\) after the right bracket (]) when making your ODBC attribute settings. If you have defined it in the Property Manager, the backslash must be included.*

Using the INSTALLDIR Property in a VBScript Custom Action



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you need to use the **INSTALLDIR** property in a VBScript custom action, use the **Property** property of the Session object. This object is accessible in every VBScript custom action.

Following is sample code:

```
szInstallDir = Session.Property("INSTALLDIR")
```

For more information, see Session Object in the Windows Installer Help Library.

Using Action Text



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

To keep end users informed, installations commonly display text on the progress dialog to describe the installation's current activity. This usually accompanies the progress bar as a means of installation status. As each standard action and custom action is encountered, a message about the action is displayed on the progress dialog. This may be especially useful for actions that take a long time to execute. The same action text is also written to the installation's log file if one is created.

An action can also send action data that needs to be processed to the Windows Installer; once the data is processed, it can be displayed on the progress dialog.

For example, when the InstallFiles action is executing, the action text "Copying new files" is displayed by default on the progress dialog. This action text is also written to the installation's log file. If you want to provide additional detailed information about what is occurring as actions are encountered, you can add a control to the progress dialog to show action details; the control would need to subscribe to the ActionData event. For the InstallFiles action, this could include the name, directory, and size of each file as it is being installed on the target system.



***Project** • InstallScript MSI installations cannot display action data on the status dialog. However, the template is written to the installation's log file.*

Specifying an Action Description and a Template for Action Data



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

InstallShield includes default action descriptions for standard actions and built-in InstallShield custom actions. InstallShield also includes default templates for many of those actions where appropriate. The templates indicate the format that should be used for the action data.

You may want to modify the default descriptions or templates of the standard actions and built-in InstallShield custom actions in your project. In addition, you may want to specify action text and action data format for your own custom actions.



Task ***To specify a description and an action data template for an action in your project:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Right-click the **Action Text** explorer and click **New**. InstallShield adds a new action text item.
3. For the name of the action text item, enter the name of the action.

   ***Important** • The names of the action text items under the Action Text explorer should match the names of standard and custom actions that are in your project. If you change the name of a custom action, you must also change the name of its action text item; otherwise, the action's text is not displayed at run time or written to the installation's log file. Note, however, that you are not required to create action text for every action in your project.*
4. In the right pane, specify the description and the template. For more information, see Action Text Settings.



***Tip** • If you want to edit the description or template of an action that is already listed under the Action Text explorer, select the action and then specify the appropriate values in the right pane.*

Displaying Action Descriptions on the Progress Dialog



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

In InstallScript MSI installations, the action text is automatically displayed on the progress dialog (that is, the STATUSEX dialog) if the dialog is enabled through a call to Enable (STATUSEX). This dialog is not available for editing.

By default, if a description has been specified for an action in your installation, that description is displayed above the progress bar on the progress dialog. Therefore, unless you have made some changes to this dialog in your project, you do not need to perform the following steps.



Task ***To display action descriptions on the progress dialog:***

1. On the **SetupProgress** dialog, add a control that will display the action description:
   a. In the View List under **User Interface**, click **Dialogs**.
   b. In the **Dialogs** explorer, expand the **All Dialogs** folder, and then expand the **SetupProgress** dialog.
   c. Click a language under the **SetupProgress** dialog. The Dialog Editor in the center pane shows the dialog in the selected language.
   d. In the **Controls** toolbar, click the **Text Area** control.
   e. Draw a rectangle on the dialog in the location where you want the action description to be displayed.
2. With the control selected, configure its settings in the right pane:
   a. For the **(Name)** setting, enter the following:

      ActionText

   b. In the **Text** setting, delete the value.
3. Add a subscription for the ActionText event to the new control:
   a. In the **Dialogs** explorer under the **SetupProgress** dialog, click the **Behavior** item. The grid in the center pane shows all of the controls on the SetupProgress dialog.
   b. Select the **ActionText** control that you just created, and then click the **Subscriptions** tab in the lower-right pane.
   c. In the upper-right pane, add a record with the following information: For the **Event** field, select **ActionText**. For the **Attribute** field, enter **Text**.

At run time, if the installation is run with a full or reduced user interface, the progress dialog shows the description of each action as it is encountered. If a description has not been defined for an action, no description is displayed on the progress dialog when that action is launched.

Displaying Action Data on the Progress Dialog



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

In InstallScript MSI installations, the action data cannot be displayed on the progress dialog (that is, the STATUSEX dialog). However, you can call Enable (INDVFILESTATUS) in your InstallScript to achieve a similar result. If you call the INDVFILESTATUS constant with the ***Enable*** *function, the STATUSEX dialog shows the fully qualified file name of each file as it is transferred when* ***FeatureMoveData****,* ***CopyFile****, or* ***XCopyFile*** *is called and the progress indicator is enabled. Note that if a template is specified for an action in an InstallScript MSI project, the template is written to the installation's log file.*

If you want to provide detailed information about what is occurring as actions are launched at run time, you can add a control to the progress dialog to show action details; the control would need to subscribe to the ActionData event. For example, when the InstallFiles action is executing, the control that shows the action details could include the name, directory, and size of each file as it is being installed on the target system.



Task ***To display the action data on the progress dialog:***

1. On the **SetupProgress** dialog, add a control that will display the action details:
   a. In the View List under **User Interface**, click **Dialogs**.
   b. In the **Dialogs** explorer, expand the **All Dialogs** folder, and then expand the **SetupProgress** dialog.
   c. Click a language under the **SetupProgress** dialog. The Dialog Editor in the center pane shows the dialog in the selected language.
   d. In the **Controls** toolbar, click the **Text Area** control.
   e. Draw a rectangle on the dialog in the location where you want the action description to be displayed.
2. With the control selected, configure its settings in the right pane:
   a. For the **(Name)** setting, enter the following:

      ActionData

   b. In the **Text** setting, delete the value.
3. Add a subscription for the ActionData event to the new control:
   a. In the **Dialogs** explorer under the **SetupProgress** dialog, click the **Behavior** item. The grid in the center pane shows all of the controls on the SetupProgress dialog.
   b. Select the **ActionData** control that you just created, and then click the **Subscriptions** tab in the lower-right pane.
   c. In the upper-right pane, add a record with the following information: For the **Event** field, select **ActionData**. For the **Attribute** field, enter **Text**.

At run time, if the installation is run with a full user interface, the progress dialog shows the details of each action as it is launched. If a template has not been defined for an action, the details are not displayed on the progress dialog when that action is launched.

Removing an Action Description or a Template for Action Data

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Task ***To remove a description or an action data template for an action in your project:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Action Text** explorer, select the action whose description or template you want to delete.
3. In the **Description** setting or the **Template** setting, delete the value.

Note *• If you want neither the action description or the action template to be used, you can delete the action text record from your project: In the Action Text explorer, right-click the action text item that you want to remove, and then click Delete.*

Note that the action text for the following actions cannot be deleted:

- *GenerateScript*
- *Rollback*
- *RollbackCleanup*

In addition, the value of the Template setting for each of these actions must always be **[1]**.

Defining Sequences

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Defining the sequences of an installation is an important part of developing an installer package. Sequences specify the order in which Windows Installer launches the standard and custom actions that control the installation process.

Project *• In Basic MSI and Transform projects, sequences also specify the order in which the dialogs are displayed.*

In InstallScript MSI projects, the user-interface dialogs are defined in the InstallScript code, and the InstallScript engine controls the user interface part of the installation.

The Custom Actions and Sequences view in InstallShield is where you define the sequences of your project. The Sequences explorer in this view lists all of the actions and dialogs in your project in chronological order, according to when they are launched during the applicable sequence. Each action and dialog is given a number in the sequence, and Windows Installer runs the sequence from the lowest number to the highest number.

Rather than having to manually provide a numeric value for every custom action and dialog that you add to your project, you can use the Custom Actions and Sequences view to insert actions or dialogs into a sequence or edit the sequence timeline. Refer to this section of the help for information about sequences.



***Project** • If you create a custom action or a custom dialog in a merge module, you need to first import that module into an installation project and then add it to a sequence through the Custom Actions and Sequences view.*

Installation Sequence



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The installation sequence is the series of actions that are executed when the installation runs in the default installation mode, such as when an end user double-clicks a new .msi file. These actions are broken down into two types:

- User Interface
- Execute



***Project** • The User Interface dialogs described below are those for a Basic MSI project. For an InstallScript MSI project, your InstallScript code performs the user interface of the installation.*

An InstallScript MSI project automatically includes the ISVerifyScriptingRuntime custom action. For details, see ISVerifyScriptingRuntime.

User Interface

The User Interface sequence contains all of the actions and dialogs needed to support a full user interface. This sequence is skipped if an installation is run in silent mode.

For complete technical details about each standard action, see the Standard Actions Reference in the Windows Installer Help Library.

Table 4-18 • User-Interface Actions and Dialogs in the Installation Sequence

Name of Action or Dialog	Type of Event	Description
SetupCompleteError	Dialog	This dialog is displayed at the end of an installation if that installation was terminated because of a fatal error.
SetupInterrupted	Dialog	This dialog is displayed at the end of an installation that was ended by the user.
SetupCompleteSuccess	Dialog	This dialog is displayed at the end of a successful installation.
ISSetupFilesExtract	Custom action	This custom action extracts any files that you have added in the Support Files view. For more information, see Using Support Files.
ISSetAllUsers	Custom action	The ISSetAllUsers custom action is inserted in the both the User Interface and Execute installation sequences only if one or more records are in the **Upgrade** table.
AppSearch	Standard action	This action can be used to identify and locate files on the target system. The Signature table that this action requires is available in the Direct Editor. Knowledge Base article Q103147 provides detailed information on using this action.
LaunchConditions	Standard action	This action evaluates a set of conditions that must return True if the installation is to continue. If any conditions fails, the user is presented with an error message and the installation ends. Launch conditions can be edited in the General information view.
SetupInitialization	Dialog	This dialog is displayed while the setup is preparing to begin. The default text for this dialog is "Preparing to install..."
FindRelatedProducts	Standard action	When this action is executed, the installer compares each installed product's upgrade code to that listed in the package's Upgrade table. If a match is found, the installed package's product code is added to the ActionProperty column of the Upgrade table.
CCPSearch	Standard action	The CCPSearch action allows you to check the end user's system for products qualifying for upgrade. The Signature table that this action requires is available in the Direct Editor.

Table 4-18 • User-Interface Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
RMCCPSearch	Standard action	The RMCCPSearch action allows you to check an end user's system for products qualifying for competitive upgrade. The Signature table that this action requires is available in the Direct Editor.
ValidateProductID	Standard action	Use the Validate Product ID action to set the ProductID property to the complete product identifier. This validation allows you to protect your software from illegal use by requiring users to enter the product ID.
CostInitialize	Standard action	This action is the first step in determining how much disk space is required by the current configuration of the installation. In order to complete costing, you need to use the CostInitialize action in conjunction with the CostFinalize action.
FileCost	Standard action	The FileCost action determines how much disk space is required by the current configuration of the installation. It checks to see if any files will be overwritten with later versions, and it calculates how overwriting those files affects disk space. To complete costing, you need to use the CostInitialize action in conjunction with the CostFinalize action.
IsolateComponents	Standard action	This action installs a component—usually a DLL file—into an isolated location so that it is used by only your application.
setUserProfileNT	Custom action	This custom action initializes the USERPROFILE directory identifier.
SetAllUsersProfileNT	Custom action	This custom action initializes the ALLUSERSPROFILE directory identifier.
setAllUsersProfile2K	Custom action	This custom action initializes the ALLUSERSPROFILE directory identifier.
ResolveSource	Standard action	Finds the location of the source and sets the SourceDir property.
CostFinalize	Standard action	The Calculate Disk Space action determines the total amount of disk space required by the installation in its current configuration. This action also verifies that all target directories are writable. The actions Initialize Disk Space Calculations and Initialize Dynamic Disk Space Calculations must be called before the Calculate Disk Space action; otherwise, the action will fail.

Table 4-18 • User-Interface Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
SetARPReadme	Custom action	The SetARPReadme custom action resolves the directory identifier used in the Read Me setting in the General Information view. This custom action is required because ARPREADME is a Windows Installer property and these properties do not format automatically.
MigrateFeatureStates	Standard action	This action is used during application upgrade. The feature states of the original installation are read from the target machine and applied to the upgraded features.
PatchWelcome	Dialog	The PatchWelcome dialog is displayed when a patch package is applied with a full user interface. It includes control events to set **REINSTALL** and **REINSTALLMODE** with the correct options. However, when the user interface is suppressed, you must set the properties at the command line.
InstallWelcome	Dialog	The InstallWelcome dialog displays your Welcome panel when an InstallScript MSI installation is run.
SetupResume	Dialog	This dialog opens when a previously canceled installation is resumed.
MaintenanceWelcome	Dialog	This dialog opens when the end user tries to change a program's installed features, remove the program, reinstall the program, run an installation for the second time, or select the current product in Add or Remove Programs.
SetupProgress	Dialog	This dialog displays the installation's progress.
ExecuteAction	Standard action	This action queries the **EXECUTEACTION** property to determine which top-level action should be called first, and then calls that top-level action. Top-level actions include the INSTALL, ADVERTISE, and ADMIN actions. Typically, this action starts the Installation Execute sequence.
ISSetupFilesCleanup	Custom action	This custom action appears when you add a file in the Support Files view. For more information, see Using Support Files.

Execute

The Execute sequence contains all of the actions that can change the machine's state and do not rely upon the user interface in order to function properly. These actions include file transfer, publishing components and features, and registering COM servers. Most of the Execute sequence is skipped when you run your installation in test mode, except for custom actions that have been inserted into the sequence.

Table 4-19 • Execute Actions and Dialogs in the Installation Sequence

Name of Action or Dialog	Type of Event	Description
ISSetupFilesExtract	Custom action	This custom action extracts any files that you have added in the Support Files view. For more information, see Using Support Files.
ISSetAllUsers	Custom action	The ISSetAllUsers custom action is inserted in the both the User Interface and Execute installation sequences only if one or more records are in the **Upgrade** table.
AppSearch	Standard action	This action can be used to identify and locate earlier versions of your product. The **Signature** table that this action requires is available in the Direct Editor. Knowledge Base article Q103147 provides detailed information on using this action.
LaunchConditions	Standard action	This action evaluates a set of conditions that must return True if the installation is to continue. If any condition fails, the installation displays an error message and the installation ends. You can edit launch conditions in the General Information view.
FindRelatedProducts	Standard action	When this action is executed, the installer compares each installed product's upgrade code to that listed in the package's **Upgrade** table (supported in the Direct Editor). If a match is found, the installed package's product code is added to the **ActionProperty** column of the **Upgrade** table.
CCPSearch	Standard action	The CCPSearch action enables you to check the end user's system for products qualifying for upgrade. The **Signature** table that this action requires is available in the Direct Editor.
RMCCPSearch	Standard action	The RMCCPSearch action enables you to check an end user's system for products qualifying for competitive upgrade. The **Signature** table that this action requires is available in the Direct Editor.

Table 4-19 • Execute Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
ValidateProductID	Standard action	Use the ValidateProductID action to set the ProductID property to the complete product identifier. Validation allows you to protect your software from illegal use by requiring end users to enter the product ID.
CostInitialize	Standard action	This action is the first step in determining how much disk space is required by the current installation configuration.
FileCost	Standard action	This action determines how much disk space will be required by the current installation configuration. This action checks to see if any files will be overwritten with newer versions and calculates how overwriting those files will affect disk space.
IsolateComponents	Standard action	This action installs a component—usually a DLL file—into an isolated location so that it will be used by only your application.
CostFinalize	Standard action	The CostFinalize action determines the total amount of disk space required by the current installation configuration. This action also verifies that all target directories are writable.
SetARPINSTALLLOCATION	Custom action	The SetARPINSTALLLOCATION custom action sets the value of the **ARPINSTALLLOCATION** property to the fully qualified path for the application's primary folder.
SetODBCFolders	Standard action	The SetODBCFolders action checks for existing ODBC drivers on the target system and sets the target directory of each new driver to the location of an existing driver.
MigrateFeatureStatus	Standard action	This action is used during application upgrade. The feature states of the original installation are read from the target machine and applied to the upgraded features.
InstallValidate	Standard action	The InstallValidate action determines if there is enough disk space available for the current installation configuration.
RemoveExistingProducts	Standard action	The RemoveExistingProducts action performs a silent uninstallation of any products whose code appear in OLDPRODUCTS.
InstallInitialize	Standard action	The InstallInitialize action signals the beginning of the actions that make changes to the end user's system.

Table 4-19 • Execute Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
AllocateRegistrySpace	Standard action	The installer makes sure that the system has at least as much free registry space available as specified in the AVAILABLEFREEREG property when it performs this action.
ProcessComponents	Standard action	The ProcessComponents action is responsible for unregistering and registering components. This action also registers or unregisters a component's key path and any other clients that the component has.
UnpublishComponents	Standard action	This action, called during uninstallation, unpublishes components that were published by your original installation, even if those components have been published by other applications.
MsiUnpublishAssemblies	Standard action	The MsiUnpublishAssemblies action handles uninstallation of the assembly with the operating system.
UnpublishFeatures	Standard action	The UnpublishFeatures action removes all references to features that were originally published during installation. These references include registry entries that contain selection-state and feature-component mapping information.
StopServices	Standard action	This action stops the Windows services that are configured to be stopped. For more information, see Installing, Controlling, and Configuring Windows Services.
DeleteServices	Standard action	This action removes the Windows services that are configured to be deleted. For more information, see Installing, Controlling, and Configuring Windows Services.
UnregisterComPlus	Standard action	This action unregisters COM+ applications.
SelfUnregModules	Standard action	This action unregisters files registered by data in the SelfReg table.
UnregisterTypeLibraries	Standard action	During uninstallation, this action unregisters every file in the **TypeLib** table that is marked for uninstallation. This table is populated when you create a new TypeLib through the COM Registration advanced setting.
RemoveODBC	Standard action	During uninstallation, this action queries the **ODBCDataSource** table, **ODBCTranslator** table, and **ODBCDriver** table to find which ODBC resources should be removed during uninstallation. The resources that are marked for removal are uninstalled.

Table 4-19 • Execute Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
UnregisterFonts	Standard action	This action unregisters information about all the fonts that are set for uninstallation.
RemoveRegistryValues	Standard action	The RemoveRegistryValues action removes values from the end user's registry if all of the following conditions are met: • The values have been authored into the **Registry** table. • The values are marked for uninstallation. • The component to which the registry entry belongs is set to run from source or is installed locally.
UnregisterClassInfo	Standard action	This action manages system registry information removal for COM classes that belong to features that are being uninstalled.
UnregisterExtensionInfo	Standard action	This action unregisters all extension-related information from the end user's system during uninstallation.
UnregisterProgIdInfo	Standard action	This action unregisters all of the ProgIDs created in the File Types advanced setting.
UnregisterMIMEInfo	Standard action	This action queries the **MIME** table of the current feature that is being uninstalled, and unregisters the MIME information for the servers found therein. This information is created through the File Types advanced setting.
RemoveIniValues	Standard action	This action removes only the .ini information that has been associated with a component in the **IniFile** table—or using the INI File Changes view. After verifying the presence in the **IniFile** table, this action removes all .ini files that are listed in the **RemoveIniFile** table if the component with which those files are associated is marked for uninstallation and the component was installed locally or set to run from source. The RemoveIniValues action also removes all .ini files that were written with the WriteIniValues action if the components with which they are associated are marked to be uninstalled.
RemoveShortcuts	Standard action	This action removes all advertised shortcuts to features that are marked for uninstallation. It also removes all non-advertised shortcuts to components that are marked for uninstallation.

Table 4-19 • Execute Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
RemoveEnvironmentStrings	Standard action	When a component is removed, this action reverses any changes made to environment variables by the WriteEnvironmentStrings action during installation or reinstallation. You can specify environment variable changes using the Environment Variables view.
RemoveDuplicateFiles	Standard action	This action removes files that were created by the DuplicateFiles action. For this action to succeed, the component associated with the duplicate file must be marked for uninstallation.
RemoveFiles	Standard action	This action removes files that were originally installed by the InstallFiles action, if those files were set to run from source or installed locally, and the component with which they are associated is marked for uninstallation.
RemoveFolders	Standard action	The Remove Folders action unregisters and removes any empty folders that are associated with components that are marked for uninstallation and run from source.
CreateFolders	Standard action	This action creates empty folders for components that are set to be installed locally. These new folders are then registered with the associated component GUID (found in the component's Component Code property).
MoveFiles	Standard action	This action enables you to move or copy files that already exist on the target system. The **MoveFiles** table, used by this action, is available in the Direct Editor.
InstallFiles	Standard action	The InstallFiles action copies all of the selected feature's files to the target machine if the component feature to which those files belong is marked for installation. Only files that are associated with components that are to be installed locally are copied to the target machine.
PatchFiles	Standard action	This action queries the **Patch** table to determine which installed files have patches available. Those files undergo the bit-wise patching process.
DuplicateFiles	Standard action	The DuplicateFiles action creates copies of certain files installed with the InstallFiles action. These files can be copied to the same directory as the original and given a different name, or they can be copied to a separate directory while maintaining the original name. The **DuplicateFiles** table, used by this action, is available in the Direct Editor.

Table 4-19 • Execute Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
BindImage	Standard action	Writes the virtual address of imported DLL file functions in the file's import address table, as specified in the **BindImage** table, which is available in the Direct Editor.
CreateShortcuts	Standard action	This action creates the shortcuts that you specified in the Shortcuts explorer in the Shortcut view or the Setup Design view.
RegisterClassInfo	Standard action	This action registers all of the COM class info that you specified in the COM Registration advanced setting, or which was extracted by the Component Wizard or a component's COM Extract at Build setting.
RegisterExtensionInfo	Standard action	The RegisterExtensionInfo action registers all the extensions that you specified in the File Types advanced setting.
RegisterProgIdInfo	Standard action	This action registers all of the ProgIDs that you defined in the Advanced Settings and that are linked to class servers or extension servers marked for installation.
RegisterMIMEInfo	Standard action	This action registers all of the MIME types that you defined in the File Types advanced setting that are linked to class servers or extension servers marked for installation.
WriteRegistryValues	Standard action	This action writes registry data to the target system if the associated component is marked for installation and set to run from source or installed locally. This registry information is the same data that you created in the Registry view.
WriteIniValues	Standard action	This action writes information to .ini files if the component with which this action is associated is set to be installed locally or run from source. This action and its corresponding tables are exposed in the INI File Changes view.
WriteEnvironmentStrings	Standard action	When a component is installed, this action changes the environment variables on the system specified in the **Environment** table or the Environment Variables view.
RegisterFonts	Standard action	The RegisterFonts action registers any fonts that you included in your installation. In most cases, they were included using the Component Wizard.
InstallODBC	Standard action	This action installs all of the drivers, translators, and data sources for the ODBC resources that you specified through the ODBC Resources view.

Table 4-19 • Execute Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
RegisterTypeLibraries	Standard action	This action registers any type libraries that you may have created in your installation, either through the COM Registration advanced setting or the Component Wizard.
SelfRegModules	Standard action	This action registers self-registering modules listed in the **SelfReg** table. This action is performed with the default user privileges.
RegisterComPlus	Standard action	This action registers COM+ applications.
InstallServices	Standard action	This action installs the Windows services that are configured to be installed. For more information, see Installing, Controlling, and Configuring Windows Services.
MsiConfigureServices	Standard action	This action configures extended customization options for Windows services. For more information, see Installing, Controlling, and Configuring Windows Services. ***Note*** • *This action is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*
StartServices	Standard action	This action starts all services that are configured to be started. For more information, see Installing, Controlling, and Configuring Windows Services.
RegisterUser	Standard action	This action registers user information to identify the user of the program.
RegisterProduct	Standard action	This action registers the product with the installer and saves the installer database on the target machine.
PublishComponents	Standard action	This action publishes all components that are associated with advertised features.
PublishFeatures	Standard action	This action registers each feature's installation state. This state can be absent, advertised, or installed. If the feature is installed, the PublishFeatures action writes the feature-component relationship to the registry.
PublishProduct	Standard action	This action publishes a product if it is being advertised.

Table 4-19 • Execute Actions and Dialogs in the Installation Sequence (cont.)

Name of Action or Dialog	Type of Event	Description
ScheduleReboot	Standard action	Insert the ScheduleReboot action into the action sequence to prompt the end user to reboot the system at the end of an installation. The ScheduleReboot action is typically placed at the end of the sequence.
InstallFinalize	Standard action	This action is the last transacted step.
RemoveExistingProducts	Standard action	This action loops through all the product codes listed in the **Upgrade** table and removes those products.
ISSetupFilesCleanup	Custom action	This custom action appears when you add a file in the Support Files view. For more information, see Using Support Files.

Advertisement Sequence



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Advertisement sequence is the list of actions that occur when an end user launches your installation with the `/j` command-line option for **MsiExec.exe**. The built-in actions in this sequence are described in the table below.

User Interface

Validation rule ICE78 requires the Advertisement User Interface sequence to be empty.

Execute

The Execute sequence contains all the actions that do not rely upon the user interface in order to function properly. These actions include file transfer, publishing components and features, and registering COM servers.

For complete technical details about each standard action, see the Standard Actions Reference in the Windows Installer Help Library.

Table 4-20 • Execute Actions in the Advertisement Sequence

Action Name	Type of Event	Description
CostInitialize	Standard action	The first step in determining how much disk space is required by the current configuration of the installation.

Table 4-20 • Execute Actions in the Advertisement Sequence (cont.)

Action Name	Type of Event	Description
CostFinalize	Standard action	Determines the total amount of disk space required by the installation in its current configuration.
InstallValidate	Standard action	Determines if there is enough disk space available for the current configuration.
InstallInitialize	Standard action	Marks the beginning of the actions that make changes to the end user's system.
CreateShortcuts	Standard action	Creates the shortcuts specified in the Shortcuts view or the Setup Design view.
RegisterClassInfo	Standard action	Registers all of the COM class information specified in the COM Registration advanced setting.
RegisterExtensionInfo	Standard action	Registers all of the extensions specified in the File Types advanced setting.
RegisterProgIdInfo	Standard action	Registers all of the ProgIDs that defined in the COM Registration advanced setting if its component is being advertised.
RegisterMIMEInfo	Standard action	Registers all of the MIME types that are defined in the File Types advanced setting if its component is being advertised.
RegisterTypeLibraries	Standard action	Registers any type libraries that were created in the installation, either through the COM Registration advanced setting or the Component Wizard.
PublishComponents	Standard action	Publishes all components that are associated with features that are being advertised.
MsiPublishAssemblies	Standard action	Manages the advertisement of common language run-time assemblies and Win32 assemblies. The action queries the MsiAssembly table to determine which assemblies have features being advertised or installed to the global assembly cache and which assemblies have a parent component being advertised or installed to a location isolated for a particular application.
PublishFeatures	Standard action	Registers each feature's installation state. This state can be absent, advertised, or installed. If the feature is installed, this action writes the feature-component relationship to the registry.
PublishProduct	Standard action	Publishes a product if it is being advertised.

Table 4-20 • Execute Actions in the Advertisement Sequence (cont.)

Action Name	Type of Event	Description
InstallFinalize	Standard action	The final step of the installation or uninstallation.

Administration Sequence



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Administration sequence is the list of actions that execute when a user launches your setup program with the / a command-line option. The built-in actions and dialogs for this sequence are defined in the tables below.

User Interface

The User Interface sequence contains all of the actions and dialogs needed to support a full user interface. This sequence is skipped if an installation is run in silent mode.



Project • *The User Interface actions described below are those for a Basic MSI project. For an InstallScript MSI project, your InstallScript code performs the user interface of the installation.*

For complete technical details about each standard action, see the Standard Actions Reference in the Windows Installer Help Library.

Table 4-21 • User-Interface Actions in the Administration Sequence

Action Name	Type of Event	Description
SetupCompleteError	Dialog	This dialog is displayed at the end of an installation if it was terminated because of a fatal error.
SetupInterrupted	Dialog	This dialog is displayed at the end of an installation that was ended by the end user.
SetupCompleteSuccess	Dialog	This dialog is displayed at the end of a successful installation.
SetupInitialization	Dialog	This dialog is displayed while the installation is preparing to begin. The default text for this dialog is "Preparing to install..."

Table 4-21 • User-Interface Actions in the Administration Sequence (cont.)

Action Name	Type of Event	Description
CostInitialize	Standard action	This action is the first step in determining how much disk space is required by the current configuration of the installation.
FileCost	Standard action	This action determines how much disk space will be required by the current configuration of the installation. This action checks to see if any files will be overwritten with newer versions, and it calculates how overwriting those files will affect disk space.
CostFinalize	Standard action	The CostFinalize action determines the total amount of disk space required by the installation in its current configuration. This action also verifies that all target directories are writable.
AdminWelcome	Dialog	This is the first dialog displayed during the Administration sequence.
SetupProgress	Dialog	This dialog displays the progress of the installation.
ExecuteAction	Standard action	This action runs the Administration Execute sequence.

Execute

The Execute sequence contains all the actions that do not rely upon the user interface in order to function properly. These actions include file transfer, publishing components and features, and registering COM servers.

Table 4-22 • Execute Actions and Dialogs in the Administration Sequence

Action Name	Type of Event	Description
CostInitialize	Standard action	This action is the first step in determining how much disk space is required by the current installation configuration.
FileCost	Standard action	This action determines how much disk space will be required by the current installation configuration. This action checks to see if any files will be overwritten with newer versions and calculates how overwriting those files will affect disk space.
CostFinalize	Standard action	The CostFinalize action determines the total amount of disk space required by the installation in its current configuration. This action also verifies that all target directories are writable.
InstallValidate	Standard action	The InstallValidate action determines if there is enough disk space available for the current setup configuration.

Table 4-22 • Execute Actions and Dialogs in the Administration Sequence (cont.)

Action Name	Type of Event	Description
InstallInitialize	Standard action	The InstallInitialize action marks the beginning of the actions that make changes to the end user's system.
InstallAdminPackage	Standard action	This action copies the installation database to the administrative installation point.
InstallFiles	Standard action	The InstallFiles action copies all of the files to the target machine if the feature that those files belong to is slated for installation. Only files that are associated with components that are to be installed locally will be copied to the target machine.
InstallFinalize	Standard action	This action is the final step of the installation or uninstallation.

User Interface Sequence



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The User Interface sequence contains all of the actions and dialogs required for the default user interface. These actions and dialogs do not make changes to the target system. Generally, they gather information about the system environment and the end user for use later in the installation.

By default, the Administration and Installation sequences each contain a User Interface sequence.



Project • *InstallScript custom actions are not supported in the User Interface sequence for InstallScript MSI projects.*

Execute Sequence



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Execute sequence is executed after the User Interface sequence, unless your installation is run in silent mode. The Execute sequence contains all the standard and custom actions that change the target system. For example, file transfer is handled during the Execute sequence. Typically, this sequence is the part of the installation where changes are made to the target machine.

Each sequence (Installation, Advertisement, and Administration) has different actions that occur in the Execute sequence. This discrepancy is due to the fact that each of those sequences achieves different goals. For example, the Installation sequence installs the product on the target machine. The Advertisement sequence installs the product's advertised shortcuts, file-type information, and COM-server data on the target machine, but it does not transfer the product's files until the end user selects the shortcut, opens a registered document, or invokes an advertised COM server. As can be gathered from the name of the sequence, the product is advertised.

Inserting Actions into Sequences



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Sequences dictate when all of the actions that are associated with an installation will be executed. You can add, remove, or reorder actions in any sequence. For example, if you want to display a readme file as part of the installation, you can add to your installation a custom action that launches the readme file. This action must be inserted into a sequence.

Inserting actions into a sequence is governed by when actions in that sequence launch. Many actions rely on other actions executing before they can function. For example, if you want to launch an executable file that will be installed as part of your installation, you cannot execute the action that relies upon that executable file until the point in the sequence when the files have been installed.



Note • *InstallShield does not provide any validation on your sequences. If an action is placed in a sequence where it cannot function properly, an error will not occur until the installation is run.*

Inserting an Action



Task ***To insert an action into a sequence:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Sequences** explorer, right-click the action or dialog that you want your action to follow and click **Insert**. The Insert Action dialog box opens, providing a list of all the actions and dialogs that can be added to the sequence.
3. In the list at the top of the dialog box, select the type of action that you want to insert.
4. In the box that shows a list of actions, select the action that you want to insert.

5. Click **OK**.

InstallShield also includes drag-and-drop support that enables you to drag and drop custom actions from the Custom Actions explorer to a sequence in the Sequences explorer.

Task ***To insert a custom action into a sequence using the drag-and-drop method:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Drag the custom action from the **Custom Actions** explorer to the appropriate location in a sequence under the **Sequences** explorer. When you drop it, drop it onto the item that should be directly before it in the sequence.

Note *• A custom action cannot be called twice in the same sequence, since the custom action name is the key in the* ***CustomAction*** *table. Therefore, you cannot insert a custom action to a sequence that already contains that custom action.*

Determining Where to Insert the Action

There are several categories of custom actions that you can include in your installation: calling a standard or Windows Installer–ready DLL function, launching an executable file, running a script (JScript, VBScript, or InstallScript), setting a property or directory, and launching a second .msi installation. All types can be called at different times depending on the configuration of the custom action. For more information, see the following:

- Sequencing a Custom Action that Calls a Function in a DLL File
- Sequencing a Custom Action that Launches an .exe File
- Sequencing a Custom Action that Calls a Script
- Sequencing Custom Actions that Set Properties or Directory Properties
- Sequencing a Custom Action that Launches a Second .msi Package

Considerations for Custom Actions from Included Merge Modules

You can control the launch of custom actions in a merge module by modifying the **ModuleInstallExecuteSequence** table in the Direct Editor. When you add the merge module to your installation project, all custom actions and dialogs included in the merge module are available for you to insert in the installation's sequences through the Custom Actions and Sequences view.

Copying a Custom Action from One Sequence to Another

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

InstallShield enables you to copy a custom action in one sequence to another sequence through a drag-and-drop operation.

Task ***To copy a custom action from one sequence to another:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Sequences** explorer, find the action that you want to copy.
3. Press and hold CTRL while dragging the custom action from one sequence to another sequence, and drop it onto the action or dialog that should be directly before it.

Note *• A custom action cannot be called twice in the same sequence, since the custom action name is the key in the* ***CustomAction*** *table. Therefore, you cannot move or copy a custom action to a sequence that already contains that custom action.*

Dialogs and standard actions cannot be moved to a different type of sequence through a drag-and-drop operation.

Reordering a Sequence

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The actions and dialogs in the Sequences explorer in the Custom Actions and Sequences view are organized by chronological order, according to when they are launched. Each action and item also has a number that identifies its location in the sequence in relation to other items' sequence numbers. The items are launched in order from the lowest number to the highest number.

When you add a dialog (to Basic MSI, MSI Database, and Transform projects) or a custom action to your project, you can specify when it should be launched by adding it to the appropriate place in a sequence.

Task ***To change the order in which actions and dialogs in a sequence execute:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Do one of the following:
   - To sequence a new custom action, drag it from the **Custom Actions** explorer to the appropriate location in a sequence under the **Sequences** explorer. When you drop it, drop it onto the item that should be directly before it in the sequence.
   - To move an action or a dialog to a different point in a sequence (or from one sequence to another), drag it from the old location and drop it onto the item that should be directly before it in the sequence.

Note *• A custom action cannot be called twice in the same sequence, since the custom action name is the key in the* ***CustomAction*** *table. Therefore, you cannot move or copy a custom action to a sequence that already contains that custom action.*

Dialogs and standard actions cannot be moved to a different type of sequence through a drag-and-drop operation.

If you change a custom action after adding it to a sequence, you do not need to reassociate it with that sequence. The action that you put in the sequence is a pointer to the real action, and it is dynamically updated whenever you make changes to the action in the Custom Actions and Sequences view.

Tip *• You can also move actions and dialogs in the* ***Sequences*** *explorer using any of the following methods:*

- *Right-click the action or dialog and click* ***Move Up*** *or* ***Move Down****.*
- *Click the action or dialog and then press CTRL+SHIFT+UP ARROW or CTRL+SHIFT+DOWN ARROW.*
- *Change the position of the action or dialog by editing its Sequence Number setting, which is displayed in the grid on the right when you click the action or dialog. To move a an action or dialog to an earlier point in a sequence, give it a lower sequence number. To move it to a later point in the sequence, give it a higher sequence number.*

Removing Actions from Sequences

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Task ***To remove an action from a sequence:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Sequences** explorer, expand the sequence that contains the action that you want to remove.
3. Right-click the action and click **Remove**.

The action is removed from that sequence only, not from all sequences or from the project. You can permanently remove an action from an installation project through the Custom Actions explorer in the Custom Actions and Sequences view.

Sequencing Rollback Custom Actions

Project *• This information applies to the following project types:*

- *Basic MSI*

- *InstallScript MSI*
- *MSI Database*
- *Transform*

Any custom action that is in your installation project and that makes direct changes to the target system should have a rollback equivalent custom action. The rollback custom action can undo those changes in the event of rollback. For example, if you have a custom action that deletes files from the target system, and you do not include a rollback custom action to restore those deleted files, the computer could be left in an unstable state even after rollback has completed. Rollback custom actions behave in a similar way to deferred custom actions. That is, they do not launch when first encountered in a sequence. Instead, they are written to the rollback script, which launches only in the event of a rollback.

When inserting a rollback custom action into a sequence, note the following:

- A rollback can occur only during the Execute sequence and not at any point during the User Interface sequence. Therefore, the rollback action must be placed after InstallInitialize and before InstallFinalize in the Execute sequence.
- The rollback custom action must be sequenced before the action it rolls back. In all other instances, the Windows Installer service runs through sequences from top to bottom—the lower the sequence number, the sooner the action launches. In the rollback script, however, the service runs in the opposite direction. Therefore, to launch your rollback action when it is needed, sequence it before the action that it rolls back.

Sequencing a Custom Action that Launches an .exe File



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Launching an executable file as your custom action can be a little trickier than calling a DLL file function in terms of where that action can be inserted into the sequence. You should consider things such as when the .exe file will be launched during the installation and which sequences you want the custom action to occur in.

Executable Files Stored in the .msi Package

When you are launching a .exe file that is stored in the .msi package, you can place the custom action anywhere in the sequence.

Table 4-23 • Settings for Actions that Launch an .exe File in the .msi Package

Action Type	Location	Scheduling	Sequence
Launch an executable	Stored in .msi package	Not applicable	Anywhere in the sequence

Installed Executable Files

If your .exe file is going to be installed as part of the installation, you should follow one of two sets of rules, depending on which scheduling property (Deferred Execution or Immediate Execution) you chose. For Immediate Execution, you must insert your action after the InstallFinalize action in order for it to work properly. For Deferred Execution, you need to insert the action after the InstallFiles action and before the InstallFinalize action.

Table 4-24 • Settings for Actions that Launch an Installed .exe File

Action Type	Location	Scheduling	Sequence
Launch an executable	Installed	Immediate Execution	After InstallFinalize
Launch an executable	Installed	Deferred Execution	After InstallFiles and before InstallFinalize

Local Executable Files

If the .exe file that you would like to run is already present on the system, you will need to insert your action after the CostFinalize action is called. See the table below for a clearer view.

Table 4-25 • Settings for Actions that Launch an .exe File that Is Already Installed

Action Type	Location	Scheduling	Sequence
Launch an executable	Local	Not applicable	After CostFinalize

Sequencing a Custom Action that Calls a Function in a DLL File

If you want to sequence a custom action that calls a function in a DLL file, there are three different ways to reference the DLL file, depending on where the DLL file is located during the installation:

- The DLL file can be streamed into the .msi file, but it is not installed.
- The DLL file can be in the target system's path (applies to a standard DLL file but not a Windows Installer DLL file).
- The DLL file can be installed with the rest of the installation files.

Other considerations include scheduling. The two main scheduling choices are Immediate Execution and Deferred Execution. Immediate Execution means that Windows Installer will execute your custom action as it processes the .msi file. Deferred Execution tells the installer to queue the action and perform it in sequence in the script.

DLL Files Stored in the .msi Package

When you are calling a function from a DLL file that is stored in the .msi package, you can place the custom action anywhere in the sequence.

Table 4-26 • Settings for Actions that Call a Function in a DLL File that is stored in the .msi Package

Action Type	Location	Scheduling	Sequence
DLL Function	Stored in .msi package	Not applicable	Anywhere

DLL Files Found in the System's Path

When you are calling a function from a DLL file that exists on the target system, there are also no restrictions on where you can place the custom action in the sequence.

Table 4-27 • Settings for Actions that Call a Function in a DLL File that Is Found in the System's Path

Action Type	Location	Scheduling	Sequence
DLL Function	On target system	Not applicable	Anywhere

Installed DLL Files

If you are calling a DLL function from a file that is going to be installed to the target machine during the installation and you have it scheduled to run during Immediate Execution, you can place the action only after the InstallFinalize action. If you want the action to occur during Deferred Execution, place the action after the InstallFiles action and before the InstallFinalize action. The following table illustrates this point:

Table 4-28 • Settings for Actions that Call a Function in a DLL File that Is Already Installed

Action Type	Location	Scheduling	Sequence
DLL Function	Installed	Immediate Execution	After InstallFinalize
DLL Function	Installed	Deferred Execution	Before InstallFinalize and After InstallFiles

Sequencing a Custom Action that Calls a Script



Project • *The VBScript and JScript information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The InstallScript information applies to the following project types:

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

Using script inside your custom action enables you to perform virtually any task. For example, you can launch external applications or create registry entries.

VBScript and JScript Code

Installed with the Product

If your script file is going to be installed as part of the installation, you should follow one of two sets of rules depending on which scheduling property (Deferred Execution or Immediate Execution) you chose. For Immediate Execution, you must insert your action after the InstallFinalize action in order for it to work properly. For Deferred Execution, you need to insert the action after the InstallFiles action and before the InstallFinalize action.

Table 4-29 • Settings for Actions that Call a VBScript or JScript File that Is Installed with the Product

Action Type	Location	Scheduling	Sequence
Run VBScript or JScript Code	Installed	Immediate Execution	After InstallFinalize
Run VBScript or JScript Code	Installed	Deferred Execution	After InstallFiles and before InstallFinalize

Stored in the Binary Table

If the script file that you are calling is stored in the Binary table, you can place the action anywhere in the sequence.

Table 4-30 • Settings for Actions that Call a VBScript or JScript File that Is Stored in the Binary Table

Action Type	Location	Scheduling	Sequence
Run VBScript/ JScript Code	Temporary	Not applicable	Anywhere in the sequence

Already Present on the Target System

If the script file that you want to call is already present on the system, you need to insert your action after the CostFinalize action is called.

Table 4-31 •

Action Type	Location	Scheduling	Sequence
Run VBScript/ JScript Code	Local	Not applicable	After CostFinalize

Stored Directly in the Custom Action

If you chose to place your script directly in the custom action, you can place the action anywhere in the sequence.

InstallScript Code

InstallScript custom actions differ from JScript or VBScript custom actions because the source files for InstallScript actions are always streamed into the .msi package. Therefore, the custom action can be inserted into a sequence anywhere after the action that has 2 as its sequence number. This limitation occurs because the script engine launches during sequence number 2. Therefore, if you call your script before the script engine launches, the system is not able to process it.

If your custom action contains feature functions or setup type dialog functions, you must insert the custom action after the CostInitialize, FileCost, and CostFinalize actions.

Sequencing Custom Actions that Set Properties or Directory Properties



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you are sequencing a custom action that sets a property or directory, the primary guideline that you need to follow is to make sure that the custom action runs before that directory or property is needed. For example, if you want to launch an executable file that is present on the target machine, the installation will probably need to search for it first.

Once the installation finds the executable file, its path can be added to the **Directory** table, and the executable file can be launched. However, if you call the custom action after you launch the executable file, the executable file will not be found.

Some other sequencing and scheduling restrictions include:

- Validation rule ICE12 requires any type-35 custom action (called **Set a directory** in the Custom Action Wizard) to be sequenced after the standard CostFinalize action in the sequences.
- Similarly, ICE12 requires any type-51 custom action (called **Set a property** in the Custom Action Wizard) that sets the value of a property found in the **Directory** table to be scheduled before the standard CostFinalize action.
- Any custom action that sets a property should be scheduled for immediate execution.

Sequencing a Custom Action that Launches a Second .msi Package



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Important *• Nested installations is a deprecated feature of the Windows Installer. Applications installed with nested installations sometimes fail because they are difficult for end users to service correctly. Microsoft Corporation recommends that you avoid using nested installations and nested-installation custom actions to install products that are intended to be released to the public. To learn more, see Concurrent Installations on the MSDN Web site.*

When you are sequencing a nested-installation custom action (called **Launch another .msi package** in the Custom Action Wizard), you must place the action in the Execute sequence after the standard CostFinalize action.

Additionally, ensure that the custom action launches before any of the files contained within the installation are needed by your main installation, if applicable. For example, if you later want to launch an executable file that is installed as part of the second .msi file, ensure that the package is installed before you try to launch the executable file.

ISSetAllUsers Custom Action

Why ISSetAllUsers Appears in Your Installation Package's Sequence

If you have entered one or more records in the **Upgrade** table for your installation (through the Upgrades view), InstallShield inserts a DLL custom action called ISSetAllUsers in both the User Interface and Execute portions of the installation sequence. When your product is installed as an upgrade, the ISSetAllUsers custom action checks the value of the ALLUSERS property in the installed version.

The ALLUSERS property is indicated in the Customer Information dialog (for Basic MSI projects) and in either the **SdCustomerInformation** or **SdCustomerInformationEx** dialog (for InstallScript MSI projects) by the end user during the initial installation. The ISSetAllUsers custom action compares the value in the installed version to the value in the new version. If the values differ, then ISSetAllUsers sets the **ALLUSERS** property of the new version to match that of the installed version.

Note that for upgrades, the **ALLUSERS** property is configurable only through a custom action.

The new installation's **ALLUSERS** property must match the installed version's property in order for the FindRelatedProducts action to succeed for the upgrade installation. In addition, if the previous version is installed for only one particular user and the upgrade is installed for all users, the resulting installation is corrupted and might not uninstall properly. ISSetAllUsers eliminates these problems by resetting the **ALLUSERS** property.

How ISSetAllUsers Works

The following example illustrates how the ISSetAllUsers custom action works:

1. My Application 1.0 is installed with the **ALLUSERS** property set to *1* (the end user selected **Install for All Users** in the end-user dialog during installation).
2. My Application 2.0 is authored as an upgrade to version 1.0 and has an entry in the **Upgrade** table to upgrade version 1.0.
3. The end user installs version 2.0 on the target system as an upgrade.
4. During the installation, ISSetAllUsers checks the value of the **ALLUSERS** property in the installed version and compares it to the new version's property by doing the following:
    a. ISSetAllUsers iterates through each entry in the **Upgrade** table of version 2.0.
    b. For every upgrade code in the **Upgrade** table, ISSetAllUsers searches for the related products on the target system.
    c. If the version and language constraints in the **Upgrade** table match one of the installed products (found in step 4b), ISSetAllUsers checks the **ALLUSERS** property of the installed version.
    d. If the value of **ALLUSERS** for the installed version differs from that of the new version, ISSetAllUsers sets the new installation's **ALLUSERS** property to this value.

Note • *If no matching product is installed on the target system, ISSetAllUsers does nothing.*

Using Support Files

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Suite/Advanced UI*

Support files are files that are available on the target system only during your product's installation process. Support files are copied to a temporary directory on the target system when installation begins. The support files are deleted when the installation is complete. The support directory is a dynamic file location and might be different on every target system and even on the same system for different installation instances.

In the Support Files view (in Advanced UI, Basic MSI, InstallScript Object, and Suite/Advanced UI projects) or the Support Files/Billboards view (in InstallScript projects and InstallScript MSI projects), you can add and remove files that you want to be available on the target system only during installation.

Adding Support Files

***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Suite/Advanced UI*

Task

To add support files to your installation project:

1. In Advanced UI, Basic MSI, InstallScript Object, and Suite/Advanced UI projects: In the View List under **Behavior and Logic**, click **Support Files**.

   In InstallScript and InstallScript MSI projects: In the View List under **Behavior and Logic**, click **Support Files/ Billboards**.

2. Optionally, to create one or more subfolders for your files, in the **Support Files** explorer, right-click the Language Independent node or one of the language-specific nodes in this view, and then click **New Folder**. InstallShield adds a folder. Enter the name that you want to use for the folder.
3. In the **Support Files** explorer, click the item that should contain the support file that you are adding.
4. Right-click anywhere in the **Files** pane and then click **Insert Files**. The **Open** dialog box opens.
5. Browse to the file that you want to include. To select multiple files, hold down the CTRL key while clicking files.
6. Click **OK**.

InstallShield adds the file to the Files pane.

***Tip** • You can also drag files from Windows Explorer and drop them into the Files pane.*

Adding a License File

***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

You can add a text file containing a license agreement in the Support Files/Billboards view.

Task ***To add a license file:***

1. In the View List under Behavior and Logic, click **Support Files/Billboards**.
2. Optionally, to create one or more subfolders for your license files, in the **Support Files** explorer, right-click the Language Independent node or one of the language-specific nodes in this view, and then click **New Folder**. InstallShield adds a folder. Enter the name that you want to use for the folder.
3. In the **Support Files** explorer, click the item that should contain the license file that you are adding.
4. Right-click anywhere in the **Files** pane and then click **Insert Files**. The **Open** dialog box opens.
5. Browse to the file that you want to include. To select multiple files, hold down the CTRL key while clicking files.
6. Call the file in the szLicenseFile parameter in one of the **SdLicense*** functions in your script.

Sorting Support Files

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

You can sort the files in the Files pane of the Support Files view (in Advanced UI, Basic MSI, and Suite/Advanced UI projects) or the Support Files/Billboards view (in InstallScript projects and InstallScript MSI projects) by clicking the heading of the column by which you want to sort the files. You can sort by any of the columns.

Adding Files and Folders to the Disk1 Folder

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Disk1 node enables you to indicate files and folders that you want to go on Disk1 of your installation media. These files and folders are not automatically installed to the target system when your installation is run. Rather, you can link to the installation media from your application or from the installation.

For example, you might include a large redistributable file with your application. You may want end users to be able to access the redistributable, but you do not want to include in the application installation. If this is the case, this file can be placed in the Disk1 folder.

Task ***To add a file or folder to the Disk1 folder:***

1. In Basic MSI projects: In the View List under Behavior and Logic, click **Support Files**.

   In InstallScript and InstallScript MSI projects: In the View List under Behavior and Logic, click **Support Files/ Billboards**.
2. In the **Support Files** explorer, click **Disk1**.
3. Right-click anywhere in the **Files** pane and then click **Insert Files**. The **Open** dialog box opens.
4. Browse to the file that you want to include. To select multiple files, hold down the CTRL key while clicking files.
5. Click **OK**.

InstallShield adds the file to the Files pane.

Removing Files or Folders from the Disk1 Folder

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Task ***To remove a file or folder from the Disk1 folder:***

1. In Basic MSI projects: In the View List under Behavior and Logic, click **Support Files**.

   In InstallScript and InstallScript MSI projects: In the View List under Behavior and Logic, click **Support Files/ Billboards**.
2. In the **Support Files** explorer, click **Disk1**.
3. In the **Files** pane, right-click the file or folder and then click **Delete**.

Adding Files and Folders to the Last Disk Folder

Project *• This information applies to InstallScript projects.*

The Last Disk node enables you to indicate files and folders that you want to go on the last disk of your installation media. These files or folders are not automatically installed to the target system when your installation is run. Rather, you can link to the installation media from your application or from the installation.

For example, you might include a large redistributable file with your application. You may want end users to be able to access the redistributable, but you do not want to include in the application installation. If this is the case, this file can be placed in the last disk folder.

Task ***To add a file or folder to the last disk image folder:***

1. In the View List under Behavior and Logic, click **Support Files/Billboards**.
2. In the **Support Files** explorer, click **Last Disk**.
3. Right-click anywhere in the **Files** pane and then click **Insert Files**. The **Open** dialog box opens.
4. Browse to the file that you want to include. To select multiple files, hold down the CTRL key while clicking files.
5. Click **OK**.

InstallShield adds the file to the Files pane.

Removing Files or Folders from the Last Disk Folder

Project • *This information applies to InstallScript projects.*

Task ***To remove a file or folder from the last disk image folder:***

1. In the View List under Behavior and Logic, click **Support Files/Billboards**.
2. In the **Support Files** explorer, click **Last Disk**.
3. In the **Files** pane, right-click the file or folder and then click **Delete**.

Adding Files and Folders to the Other Disk Folder

Project • *This information applies to InstallScript projects.*

The Other node enables you to indicate files or folders that you want to go on a disk of your installation media other than the first or last disk. These files or folders are not automatically installed to the target system when your installation is run. Rather, you can link to the installation media from your application or from the installation.

Task ***To add a file or folder to a disk image folder:***

1. In the View List under Behavior and Logic, click **Support Files/Billboards**.
2. In the **Support Files** explorer, click **Other**.
3. Right-click anywhere in the **Files** pane and then click **Insert Files**. The **Open** dialog box opens.
4. Browse to the file that you want to include. To select multiple files, hold down the CTRL key while clicking files.
5. Click **OK**.

InstallShield adds the file to the Files pane.

Tip *• To specify the disk, run the Release Wizard; in the General Options panel, click the Other Disk Files button.*

Removing Files or Folders from the Other Disk Folder

Project *• This information applies to InstallScript projects.*

Task

To remove a file or folder from a disk image folder:

1. In the View List under Behavior and Logic, click **Support Files/Billboards**.
2. In the **Support Files** explorer, click **Other**.
3. In the **Files** pane, right-click the file or folder and then click **Delete**.

Removing Support Files

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Suite/Advanced UI*

Task

To remove support files from your project:

1. In Advanced UI, Basic MSI, InstallScript Object, and Suite/Advanced UI projects: In the View List under **Behavior and Logic**, click **Support Files**.

   In InstallScript and InstallScript MSI projects: In the View List under **Behavior and Logic**, click **Support Files/ Billboards**.
2. In the **Support Files** explorer, click the item that contains the support file that you want to delete.
3. In the **Files** pane, right-click the file or folder and then click **Delete**.

Defining the End-User Interface

This section of the documentation covers different features of InstallShield that enable you to define different aspects of the end-user interface. Some portions of the "Defining the End-User Interface" section discuss how to create and work with dialogs for the end-user interface of your installation. Others discuss topics such as strings, which let you localize your installation.

Working with Dialogs

This section of the documentation covers some of the basic and more advanced aspects of working with end-user dialogs for various project types in InstallShield.



Project • *In Basic MSI installations, Windows Installer typically controls the run-time user interface. In InstallScript and InstallScript MSI installations, the InstallScript engine typically controls the run-time user interface. Therefore, some portions of this section of the documentation apply to Basic MSI installations, some apply to InstallScript and InstallScript MSI installations, and some apply to all three project types.*

Working with Dialogs in Any Project Type

Dialogs provide the user interface for your installation. They request information from the end user and provide feedback about the progress of the installation process.

The way in which you work with the dialogs in your project depends on the type of installation that you are creating. Adding dialogs to an InstallScript or InstallScript MSI project requires different steps than adding dialogs to a Basic MSI project.

Common Operations

Some of the dialog operations are common to Basic MSI, InstallScript, and InstallScript MSI projects.

- Using the Dialog Wizard to Create a New Dialog
- Exporting a Dialog to an .isd File for Use in Other Projects
- Importing Dialogs from an .isd File
- Exporting All Dialogs to an .rc File
- Importing Dialogs from Resource .dll Files
- Exporting Dialogs to Other Projects

Using the Dialog Wizard to Create a New Dialog

InstallShield provides a Dialog Wizard that enables you to add a new dialog to your installation and configure it by stepping through the wizard panels.

Adding the Ability to Create or Set an Existing User Account



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Project-specific differences are noted where appropriate.

Many server applications require the specification of a user account during installation. Having a user account available is often necessary because it enables a server application to access resources that are restricted to other users. InstallShield offers support for setting an existing Windows user account or creating a new one during installation by enabling you to add the appropriate run-time dialogs to your installation.

Table 4-1 • Project-Specific Details for Adding User Account Support

Project Type	Additional Information
Basic MSI	For Basic MSI projects, the specified user name, password, and group information entered in the LogonInformation dialogs are stored in the following properties: • IS_NET_API_LOGON_USERNAME • IS_NET_API_LOGON_GROUP • IS_NET_API_LOGON_PASSWORD
InstallScript	For InstallScript projects, the specified user name, password, and group information entered by the end user in the LogonInformation dialogs are stored in the following global variables: • IFX_NETAPI_USER_ACCOUNT • IFX_NETAPI_PASSWORD • IFX_NETAPI_GROUP
InstallScript MSI	For InstallScript MSI projects, the specified user account, group, and password are stored in the following global variables and properties: • IFX_NETAPI_USER_ACCOUNT • IFX_NETAPI_GROUP • IFX_NETAPI_PASSWORD • IS_NET_API_LOGON_USERNAME • IS_NET_API_LOGON_GROUP • IS_NET_API_LOGON_PASSWORD

Adding Support for the LogonInformation Dialogs to a Basic MSI Project

Task ***To add the LogonInformation dialogs to a Basic MSI project:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click **All Dialogs**, and click **New Dialog**. The **New Dialog Wizard** opens.
3. Complete each panel of the wizard. In the **Dialog Template** panel, select **Logon Information Panel and Associated Child Dialogs**.

When you are done completing each panel of the New Dialog Wizard, build and run your installation project. When the dialog sequence in which the LoginInformationDialog dialog is executed, the appropriate dialogs are displayed.

Adding Support for the LogonInformation Dialogs to an InstallScript or InstallScript MSI Project

Task ***To add support for the LogonInformation dialogs to either an InstallScript or InstallScript MSI project:***

1. Navigate to the location in your InstallScript code where you want to insert the LogonInformation dialog set. In most situations, you will add it within **OnFirstUIBefore**. For example:

   ```
   Dlg_SdLogon:
   nResult = SdLogonUserInformation(szTitle, szMsg, szAccount, szPassword);
   if (nResult = BACK) goto Dlg_SdWelcome;
   ```

2. In InstallScript MSI projects, add the following function so that the Windows Installer properties are set to the same value as the InstallScript global variables. You can add it after calling **SdLogonUserInformation**.

   ```
   OnLogonUserSetMsiProperties();
   ```

3. On the **Build** menu, click **Settings**. The **Settings** dialog box opens.
4. Click the **Compile/Link** tab.
5. In the **Libraries (.obl)** box, enter the name of your new library file (*.obl):

   NetApiRT.obl

Note • *You should add the new library file name before the* **isrt.obl** *file name.*

Once you have added the InstallScript code, build and run your installation. When the script is executed, the appropriate dialogs are displayed.

Limitations of the LogonInformation Dialogs in Basic MSI, InstallScript, and InstallScript MSI Projects

When an end user attempts to use the LogonInformation dialogs, they may encounter a blank list of domains, or they may encounter a “Server not found” error. The following scenarios may cause this behavior:

- The target system is not on a domain.
- The Computer Browser service is not enabled on the target system.

- A firewall is configured without an exception for the Computer Browser service.
- NetBIOS over TCP/IP is not enabled on the target system.
- No master browser server is configured on the network.
- Broadcast traffic is disabled on the network.

Exporting a Dialog to an .isd File for Use in Other Projects

If you want to reuse an end-user dialog in other projects, you can export the dialog as an .isd file and then import it into other projects as needed. The .isd file contains the behavior and layout information for the dialog.

Task ***To export a dialog to an .isd file:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click the dialog and then click **Export to Dialog File**. The **Save As** dialog box opens.
3. Browse to the folder to which you want to save the .isd file. You can rename the file if required.
4. Click **OK**.

You can now use the .isd file to import the dialog into another installation project.

Importing Dialogs from an .isd File

InstallShield enables you to import a dialog into your installation project. The file for the dialog (.isd file) can be stored in a repository, or it can be stored in some other location.

Task ***To import a dialog into your project from an .isd file:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click the dialog and then click **Import Dialog**. The Import Dialog dialog box opens.
3. Do one of the following:
   - In the **Repository Items** box, click the dialog that you want to add to your project.
   - If the file for the dialog box (.isd file) that you want to import is not stored in the repository, click the **Browse** button to select it.
4. Click **OK**.

InstallShield adds a copy for each supported language to your project.

Note *• When you import a dialog, make sure that any string identifiers, properties, and actions referenced by the dialog are present in the new installation project.*

Resolving Conflicts

When you import a dialog, InstallShield checks to make sure that the name of the dialog and its controls are unique—as required by Windows Installer. You cannot import a dialog with the same name as an existing dialog because InstallShield assumes that you are trying to import an identical dialog.

If you try to import an .isd file that uses the names of controls or string identifiers that already exist in your installation project, InstallShield asks how you want to resolve those conflicts. You can either overwrite the existing values or skip the imported values in order to leave the existing ones.

Displaying Dialogs During Installation

Adding a dialog to a project does not mean that the dialog will be displayed in the installation. For information on inserting the dialog into one or more of the project's sequences, see Displaying Dialogs During Basic MSI Installations.

Publishing a Dialog (.isd) File to a Repository

If you have an existing dialog (.isd) file that you would like to reuse in other projects or share with other users, you can publish it to a repository.



Task ***To publish a dialog to a repository:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click the dialog and then click **Publish Wizard**. The **Publish Wizard** opens.
3. Complete the panels in the Publish Wizard.

After you have imported a dialog from a repository into a project, there is no link between the current dialog and the existing repository dialog. If you make a change to the dialog and then republish it to the repository, it does not affect the dialog in the project to which it was imported. However, you can reimport the dialog from the repository into your project.

Exporting All Dialogs to an .rc File

If you would like to edit your dialogs' resources in a resource editor such as Visual Studio, you can export the dialogs to an .rc file.



Task ***To export a dialog to a resource script (.rc) file:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click **All Dialogs** and click **Export Dialogs to Resource Script**.

InstallShield copies all of the dialogs' resources for the default language to a file named <project name>.rc in the project location.

Every attempt is made to preserve each dialog's layout as much as possible. However, since some information in the Dialog Editor has no counterpart in a resource editor, InstallShield exports the resources as they appear in the Dialog Editor with the following exceptions:

Table 4-2 • Resources that Are Not Exported to an .rc File

Exception	Description
String entries	All strings entries are resolved to the value of the default language.
Paths	The paths, including any path variables, to any resources, such as text files or bitmaps, are preserved to make it easier to import the dialog at a later point.
Selection trees	A Windows Installer selection tree becomes a standard tree control in the .rc file.
Text style	While the Dialog Editor maintains separate properties for font information (base text style and text style) and maximum character length, these properties are grouped together as they are in the Windows Installer tables.
Behavior	Unlike .isd files, which contain complete copies of the dialog's layout and behavior, .rc files store only the resources.

Editing the Dialogs in a Resource Editor

You can edit the dialogs' geometry in a resource editor—or Notepad—as you would for any Windows dialog.

Before importing the dialogs back into your project, you must compile the .rc file into a .res file and then build it into a .dll file.

Task ***Assuming again that you use Visual Studio, follow the steps below to build MyProject.rc into MyProject.dll:***

1. Use the Resource Compiler (**Rc.exe**) to compile **MyProject.rc** into **MyProject** with the following command-line statement:

   ```
   rc MyProject.rc
   ```

2. Run the Incremental Linker (**Link.exe**) to build a .dll file with the following command:

   ```
   link /DLL /NOENTRY /NODEFAULTLIB /MACHINE:iX86 /OUT:MyProject.dll MyProject.res
   ```

Importing Dialogs from Resource .dll Files

Task ***To import all of the dialog resources from a .dll file:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click **All Dialogs** and click **Import Dialogs from Resource Dlls**. The **Open** dialog box opens.
3. Browse to the .dll file that contains the dialog resources that you want to import.

4. Click **Open**.

InstallShield adds each dialog to your project. Naming conflicts are resolved by adding a number to the imported dialog to make it unique.

If an imported control displays text that is already found as a string entry, InstallShield uses the existing string entry for the control's Text property.

***Note** • When you are importing a dialog, ensure that any properties referenced by the dialog are present in the new installation project.*

Adding a dialog to a project does not mean that the dialog will be displayed during the installation run time. For information on inserting the dialog into one or more of the installation's sequences, see Displaying Dialogs During Basic MSI Installations.

Even though dialogs can be exported to an .rc file, you must build the resources into a .dll file before you can import them into InstallShield. For more information, see Exporting All Dialogs to an .rc File.

Exporting Dialogs to Other Projects

InstallShield enables you to insert a dialog from the current project into another existing project by exporting it.

Task ***To export a dialog to an existing project:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click the dialog and click **Export to Project File**. The **Export Into** dialog box opens.
3. Browse to the location of an existing .ism file. It can be either an installation project or a merge module project, but the file cannot be open in another instance of InstallShield.
4. Click **Save**.

A copy of this dialog is added to the specified .ism file, along with every control and all of the dialog's behavior. Any string entries, properties, and path variables used in the dialog are also copied to your new project.

If the target .ism file already has a dialog of that name with different properties, the Resolve Conflict dialog box opens to offer you options for resolving the conflicting dialogs.

You can also publish dialogs to a repository and reuse them in other installation projects. For more information, see Using a Repository to Share Project Elements.

Removing Dialogs from Projects

Basic MSI Projects

Task ***To remove a dialog from a Basic MSI project:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click the dialog and click **Delete**.

Note • *Right-clicking a dialog in the Custom Actions and Sequences view and clicking Remove does not delete the dialog from your project. It only removes it from that sequence.*

InstallScript and InstallScript MSI Projects

Task ***To remove a dialog from an InstallScript or InstallScript MSI project:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click the dialog and click **Delete**.
3. In the View List under **Behavior and Logic**, click **InstallScript**.
4. Delete the script that adds the dialog to your installation's end-user interface.

Creating and Configuring Custom Dialogs

Consider the following tips when you create and configure custom dialogs in different project types.

InstallScript and InstallScript MSI Projects

When you add a new dialog in the Dialogs view, an entry for the dialog is added to the **Dialog** table. You can directly modify data in this table in the Direct Editor.

In the Direct Editor, you can specify a value for the ISResourceId field for the new dialog. This ISResourceId value is used within the script to identify this dialog.

Once you have added the dialog to your project, you need to call functions such as **EzDefineDialog** and **WaitOnDialog** to load this dialog in memory and display it at run time. The corresponding ISResourceId value in the Dialog table is used as the fourth argument of **EzDefineDialog** to reference the custom dialog. Also, in the second argument of **EzDefineDialog**, it is advantageous to specify ISUSER as opposed to a null string. The **EzDefineDialog** function should look like the following:

```
EzDefineDialog("MyCustomDialog", ISUSER, "", 12005)
```

In the above example, 12005 is the ISResourceId of this custom dialog, as specified in the Dialog table.

Custom dialog functions can then be called to manipulate the custom dialog to your needs. These functions are documented in the InstallScript Language Reference.

Basic MSI Project

When you create a custom dialog in the Dialogs view of a Basic MSI project, a Windows Installer–type dialog is created. InstallScript custom dialog functions cannot be used with this Windows Installer–type dialog. To display this dialog in your installation, it must be placed in sequence between two dialogs by using their Next and Back controls.

Undoing Changes in the Dialog Editor

The Dialog Editor remembers up to 50 actions for each dialog and allows you to restore each change starting from the most recent one. Additionally, you can undo changes even after saving the project. However, if you close a project and reopen it, the undo history is purged.

Undoing Changes



Task ***To undo a change, do any of the following:***

- Press CTRL+Z.
- On the **Edit** menu, click **Undo**.
- Click the **Undo** button on the toolbar.

Redoing Undone Changes



Task ***To place an undone change back into effect, do one of the following:***

- Press CTRL+Y.
- On the **Edit** menu, click **Redo**.
- Click the **Redo** button on the toolbar.

The Undo and Redo buttons and menu commands are enabled while you are using the Dialog Editor. When you undo an action, you move back through the history of actions you performed on that dialog. When you redo, you move forward.

For example, if you resized a button and then changed its Tab Index property, clicking Undo would restore the previous value of the Tab Index property. Clicking Undo a second time returns the button to its original size. Clicking Redo twice puts the changes back into effect on the button's size and then its tab index.



Note • *You cannot undo changes made to a string entry through the Dialog Editor.*

Working with Dialogs in InstallScript and InstallScript MSI Projects

InstallShield enables you to call specific functions in your script to display end-user dialogs. The Dialogs view contains a list of standard dialogs as well as any custom dialogs that you have added to your project.

This section of the documentation discusses how to create and perform basic functions with dialogs in InstallScript and InstallScript MSI projects.

Displaying Dialogs During InstallScript and InstallScript MSI Installations

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Most of the user interface of an InstallScript or InstallScript MSI installation is defined in event handlers such as **OnFirstUIBefore** and **OnFirstUIAfter**. The following sample InstallScript code in **OnFirstUIBefore** is for the **SdWelcome** and **SdLicense2** dialogs:

```
Dlg_SdWelcome:
    szTitle = "";
    szMsg = "";
    nResult = SdWelcome( szTitle, szMsg );
    if (nResult = BACK) goto Dlg_Start;

Dlg_SdLicense2:
    szTitle = "";
    szOpt1 = "";
    szOpt2 = "";
    szLicenseFile = SUPPORTDIR ^ "License.rtf";
    nResult = SdLicense2Ex( szTitle, szOpt1, szOpt2, szLicenseFile, bLicenseAccepted, TRUE );
    if (nResult = BACK) then
        goto Dlg_SdWelcome;
    else
        bLicenseAccepted = TRUE;
    endif;
```

Every dialog function returns a constant indicating which button the end user clicked to exit the dialog. To handle the end user clicking the Back button on a dialog, directly after the dialog is an if statement that compares the dialog's return value to the constant BACK, using a goto statement to jump to a label just before the previous dialog. In the aforementioned code example, if the end user clicks the Back button on the **SdLicense2** dialog, the goto statement jumps to the Dlg_SdWelcome label, and the **SdWelcome** dialog is displayed to the end user. Therefore, if you insert a dialog function between two dialogs such as **SdWelcome** and **SdLicense2**, you need to adjust the if statements, labels, and goto statements as appropriate.

Task ***To display a dialog to the end user as part of the installation's user interface:***

1. In the View List under **Behavior and Logic**, click **InstallScript**.
2. Add the dialog function to the appropriate script event handler.
3. Modify the dialog function's parameters according to how you want the dialog to behave.

   To learn about the available parameters, refer to the dialog function documentation:

   - Dialog Functions
   - Dialog Customization Functions
4. Use the script to direct the flow of the dialogs—for example, by using if and goto statements.

Changing the Text on Dialogs in InstallScript and InstallScript MSI Projects

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Most dialog functions accept a string argument called szTitle that defines the text appearing in the title area of the dialog. For example, the call to **SdRegisterUser** appears as follows:

```
SdRegisterUser( szTitle, szMsg, szName, szCompany );
```

By default, the parameter szTitle is defined as a null string (""), which indicates that the dialog should use the default title text. For **SdRegisterUser**, the default title appears as shown in the following screen shot.

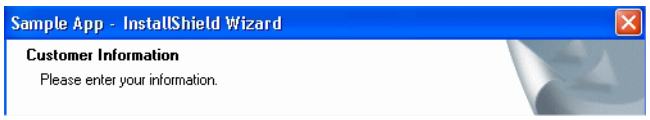


Figure 4-1: Default Title Text for the SdRegisterUser Dialog

To modify the title, you can enter specific strings to display for the `szTitle` parameter. The title is divided into two sections—the bold text at the top of the title area, and the regular text under the main title.

To specify the two sections in the value of `szTitle`, place a newline character `\n` between the two sections.

```
SdRegisterUser ("New Title\nThis is a new subtitle.",
    szMsg, szName, szCompany);
```

After recompiling the script and running the installation, the title is displayed as shown in the following sample screen shot.

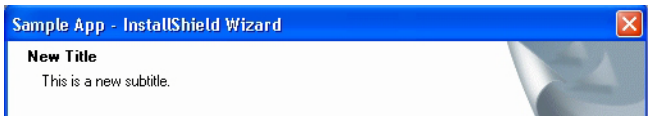


Figure 4-2: Changed Title Text for the SdRegisterUser Dialog

To change other text displayed on a dialog, there are usually one or more parameters (such as `szMsg`) that contain the text to be displayed. As with dialog box titles, a null string in a message parameter indicates that the dialog should use the default message text provided by InstallShield.

If your installation contains more than one UI language, you can use string identifiers instead of hard-coded strings in your InstallScript. For more information, see Using String Entries in InstallShield.

Background About the Default Images on Dialogs in InstallScript and InstallScript MSI Projects

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Two types of images—the default banner images on interior dialogs, as well as the computer image on the left side of the exterior skinned dialogs such as the Welcome dialog—are used for static dialog user interface controls that have a control identifier of 1200. The default images are .png images, which allow for transparency. The Text field for these controls uses the following syntax by default to identify the image:

```
@@ResourceID;ScaleFactor
```

ResourceID indicates the resource identifier number that should be used to look up either a .png image (stored as resource type PNG) or a bitmap image (stored as resource type RT_BITMAP). *ScaleFactor* indicates the DPI scale percentage for which the image is intended.

For example, the scale factor can be 100% (96 DPI), 125% (120 DPI), 150% (144 DPI), or 200% (192 DPI). If the scale factor that is specified for an image is 200, the image will be shrunk down for display on target systems that are running less than 200% DPI scaling. On a 200% target system, it will be displayed at 1:1. If the scale factor that is specified for the image is100, the image will be scaled up for display on target systems that are running 200% DPI scaling.

The previous format for identifying bitmap images (.bmp) still works, but it does not have support for scaling, or for .png images:

```
@BitmapResourceID;TransparentFlag;3-DFlag;<unused>;TransparentColorKey
```

BitmapResourceID indicates the resource identifier number for the bitmap image. *TransparentFlag* is 1 (true) or 0 (false), indicating whether the color that is specified in the *TransparentColorKey* field will be transparent when the bitmap is displayed. *TransparentColorKey* specifies an RGB value that is a transparent color for the bitmap.

Creating New Custom Dialogs in InstallScript and InstallScript MSI Projects



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

To create a custom dialog, you need to perform the following general steps:

1. Use the New Dialog Wizard to add a new custom dialog to your project. For more information, see Using the New Dialog Wizard to Add a New Custom Dialog to an InstallScript or InstallScript MSI Project.
2. Add controls to the dialog. For more information, see Adding a Control to a Dialog in an InstallScript or InstallScript MSI Project.
3. Create a script function that loads the dialog into memory, displays it on the screen, handles the end user's interaction with the dialog's controls, and closes the dialog when the user is finished with it. For more information, see Using InstallScript to Implement Custom Dialogs.

Using the New Dialog Wizard to Add a New Custom Dialog to an InstallScript or InstallScript MSI Project



Project • *This information applies to the following project types:*

- *InstallScript*

- *InstallScript MSI*



Task ***To add a new custom dialog to your project:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click **All Dialogs**, and click **New Dialog**. The **New Dialog Wizard** opens.
3. Follow the wizard panels to create the new dialog. In the **Dialog Template** panel, select the **NewScriptBasedDialog** template or the **NewSkinnableDialog** template.

   The NewScriptBasedDialog and NewSkinnableDialog templates include a hidden control that has a Control Identifier value of 2. To enable end users to cancel the installation by clicking the close button in the upper-right corner of the InstallScript dialog, the dialog must have a button control whose Control Identifier property is set to this value. The Blank Dialog template does not include this control. For more information, see Using InstallScript to Implement Custom Dialogs.

Once you have added a new custom dialog to your project, you can add dialog controls.

Editing the Layout of a Dialog in an InstallScript or InstallScript MSI Project



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

You can use the Dialog Editor to add, remove, and rearrange controls on a selected dialog. You can also use the Dialog Editor to modify the properties of the dialog or its controls. The InstallShield Dialog Editor functions much in the same way as other resource editors.

When you use the Dialog Editor to change the layout or appearance of a dialog, you are changing only the current project's copy of the dialog; the changes do not affect any other projects.



Task ***To access the Dialog Editor, do one of the following:***

- In the **Dialogs** view, click a dialog and then in the dialog preview pane on the right, click the link for the language of the dialog that you want to edit.
- In the **Dialogs** view, right-click a dialog and click **Edit**.

InstallShield displays the Dialog Editor in the center pane.



Note • *When an end user runs your installation on a Windows platform that uses a desktop display theme, the theme is used to display your installation's end-user dialogs.*

Adding a Control to a Dialog in an InstallScript or InstallScript MSI Project

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

You can use the Controls toolbar in InstallShield to add controls to a dialog.

Task ***To add a control to a dialog:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **All Dialogs** folder, and then expand the dialog item that should have the new dialog control.
3. Click a language under the dialog item that you expanded. The Dialog Editor in the center pane shows the dialog in the selected language.
4. In the **Controls** toolbar, click the control that you want to add.
5. Draw a rectangle on the dialog in the location where you want the control to be displayed.
6. In the right pane, set the control's properties.

After adding the control in the Dialog Editor, you need to use InstallScript to process the end user's interaction with the control. To learn more, see Using InstallScript to Process Dialog Controls.

Note *• For projects created in Microsoft Visual Studio, you can use the Toolbox to add dialog controls.*

Setting a Control's Properties

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The dialog itself and every control on it has a property sheet in the Dialog Editor. These properties determine such things as a control's size and position, its default text (as a string identifier), or whether the dialog is modal.

To edit one of the property sheets, select the control on the dialog, or select a control's or dialog's name from the box above the property sheet. The properties vary depending on the type of control:

- Dialog
- Check Box
- Push Button
- Edit Field

- Combo Box
- Text Area
- List Box
- Radio Button
- Bitmap
- Group Box
- Line
- Radio Button Group
- Selection Tree
- Progress Bar
- List View
- Icon

Displaying Controls on Top of a Bitmap



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

When you are setting the properties of a bitmap and other dialog controls, you can indicate that the controls should be placed on top of the bitmap.



Task ***To display controls on top of a bitmap:***

- The value for the **Tab Index** property of the bitmap must be lower than the value for the **Tab Index** property of any controls that are to be placed on top of the bitmap. The easiest way to configure this is to set the bitmap control's **Tab Index** property to 0.
- All controls—including the bitmap—must have their **Tab Stop** properties set to **True**.

Using InstallScript to Implement Custom Dialogs



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The next step in creating a custom dialog is to write an InstallScript function that processes the end user's interaction with the dialog controls.

Task ***To use InstallScript to implement a custom dialog in an InstallScript project:***

1. Begin by creating a new InstallScript source file to contain the custom dialog code. Name the file **CustomDialog.rul**.

   Tip *• To be able to use the dialog script in other projects, copy the .rul file to another location. In your other projects, open the* ***InstallScript*** *view, right-click the* ***Files*** *item in the* ***InstallScript*** *explorer, and click* ***Insert Script Files****.*

2. Inside **CustomDialog.rul**, place the prototype and implementation for your custom dialog function. Add the following code to **CustomDialog.rul**.

   ```
   prototype NUMBER CustomDialog( );
   function NUMBER CustomDialog( )
   begin
   end;
   ```

3. When you compile your script, indicate that the main script **Setup.rul** should include the code for your CustomDialog function by inserting the following line near the top of **Setup.rul**:

   ```
   #include "CustomDialog.rul"
   ```

4. In the **CustomDialog.rul** script, define the constants that store the numeric IDs of the controls that you added to the dialog. If you copied the Back, Next, and Cancel buttons from a standard dialog, you can add the following lines near the top of **CustomDialog.rul**:

   ```
   // control identifiers
   #define BUTTON_NEXT    1
   #define BUTTON_BACK   12
   ```

   In general, the numeric ID of a control on a dialog is the number listed in the control's Control Identifier property, displayed in the property list when you select the control in the Dialog Editor.

   Important *• To enable end users to cancel the installation by clicking the close button in the upper-right corner of the InstallScript dialog, the dialog must have a button control with a value of 2 for the Control Identifier property. You can place this button anywhere on the dialog, and if necessary, you can make this button invisible. This is necessary because the InstallScript engine passes the WM_CLOSE message to the script only if it comes from a valid control in the dialog. If the dialog does not have a button with this identifier, the installation does not pass through the message that is generated when the dialog's close button is clicked.*

   You need to define additional constants for every other control (for example, check box, edit field, or list box) with which the end user can interact.

5. Your CustomDialog function loads the custom dialog into memory using the **EzDefineDialog** function:

   ```
   EzDefineDialog(
     "CustomDialog", // nickname for dialog
     ISUSER,         // DLL containing the dialog's resources
     "CustomDialog", // name of dialog in Dialogs view
     0);             // numeric resource ID for dialog; not used here
   ```

   To learn which arguments you use with **EzDefineDialog**, see EzDefineDialog.



Tip • *The resource ID of a dialog is the dialog's name, as it appears in the Dialog Editor. If you need to specify a numeric resource ID, you can add a numeric ID to the* ***ISResourceId*** *column in the Dialog table for the custom dialog. You can access the Dialog table in the Direct Editor.*

6. Create a message loop in your script for the custom dialog. The message loop repeatedly calls the function **WaitOnDialog**, which returns the numeric control ID for the control with which user interacts with. A typical message loop appears as follows.

```
// repeatedly call WaitOnDialog until the user exits the dialog
// with the Next, Back, or Cancel button
while (!bDone)

// wait for the user to interact with a control,
// then return its ID
nCtrl = WaitOnDialog("CustomDialog");

// use a switch statement to handle the different controls
switch (nCtrl)

    case CONTROL1:
        // do something when user clicks CONTROL1

    case CONTROL2:
        // do something when user clicks CONTROL2

    // case statements for other controls

endswitch;

endwhile;
```

For example, CustomDialog currently contains **Next**, **Back**, and **Cancel** buttons. Its message loop might appear as follows:

```
while (!bDone)

nControl = WaitOnDialog("CustomDialog");

switch (nControl)

    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable
        // states for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg("CustomDialog");
        SdGeneralInit("CustomDialog", hwndDlg, 0, "");

    case BUTTON_BACK:
        // user clicked Back
        nReturn = BUTTON_BACK;
        bDone = TRUE;

    case BUTTON_NEXT:
        // user clicked Next
        nReturn = BUTTON_NEXT;
        bDone = TRUE;
```

```
        default:
            // check standard control handling
            if (SdIsStdButton(nControl) && SdDoStdButton(nControl)) then
                bDone = TRUE;
            endif;

    endswitch;

    endwhile;
```

7. When the end user exits the dialog by clicking **Back** or **Next**, you should remove the dialog from the screen and from memory using **EndDialog** and **ReleaseDialog**:

   ```
   EndDialog("CustomDialog");
   ReleaseDialog("CustomDialog");
   ```

   To use the dialog in the main script, add a call to the CustomDialog function in, for example, the **OnFirstUIBefore** event handler of Setup.rul.

   ```
   Dlg_SdWelcome:
       szTitle = "";
       szMsg   = "";
       nResult = SdWelcome(szTitle, szMsg);

   Dlg_CustomDlg:
       nResult = CustomDialog( );
       if (nResult = BUTTON_BACK) goto Dlg_SdWelcome;

   // etc.
   ```

   If the end user clicks **Back** or **Next**, the script displays the previous or following dialog. If the user clicks **Cancel**, the standard confirmation dialog (handled by the **OnCanceling** event handler) is displayed.



***Note** • For information on implementing dialog control functionality, see Using InstallScript to Process Dialog Controls.*

Special Messages

In addition to returning control identifiers, the **WaitOnDialog** function returns some special messages.

- Immediately before the dialog is displayed on the screen, **WaitOnDialog** returns the message constant DLG_INIT. In the DLG_INIT case statement, you can set the default states of check boxes and radio buttons, populate and set the current selection in a list box or combo box control, or set the initial text of an edit field.
- If an error occurs, **WaitOnDialog** returns the constant DLG_ERR.

Using InstallScript to Process Dialog Controls



***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

In InstallScript and InstallScript MSI installation projects, you use InstallScript to process the controls that you add to your custom dialogs.

Using Check Box Controls

In addition to handling button clicks, a custom dialog generally needs to be able to retrieve the end user's selections in dialog controls such as check boxes.



***Note** • InstallShield has a standard dialog called AskOptions. This dialog displays up to nine check boxes or radio buttons, and therefore it is not necessary to create a custom dialog to display only check boxes to the end user.*

As with push buttons, for each check box control that you add to a dialog, you generally add a #define statement to your script that assigns a symbolic name to the numeric control ID. For example, if you add a check box control to a custom dialog, the control's numeric ID will appear in the Control Identifier property of the control. If the numeric ID is 1302, you would add the following line to your script.

```
#define MYCHECKBOX1 1302
```

For most types of controls, InstallScript defines **CtrlGet** and **CtrlSet** functions with which you can get or set the current state or data for a control. For example, you can get and set the current state of a check box control using **CtrlGetState** and **CtrlSetState**. In the DLG_INIT case of your dialog's message loop, you can call **CtrlSetState** to set the initial selection state of the check box. The following code causes the check box to appear initially selected.

```
case DLG_INIT:
    // Initialize the back, next, and cancel button enable/disable states for this dialog
    // and replace %P, %VS, %VI with IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION,
    // and IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
    hwndDlg = CmdGetHwndDlg("CustomDialog");
    SdGeneralInit("CustomDialog", hwndDlg, 0, "");
    CtrlSetState("CustomDialog", MYCHECKBOX1, BUTTON_CHECKED);
```

Similarly, you can add the following code outside the dialog's message loop to detect the final selection state of the check box.

```
nState = CtrlGetState("CustomDialog", MYCHECKBOX1);
if (nState = BUTTON_CHECKED) then
    // check box selected
else
    // check box unselected
endif;
```

Using Edit Fields

You can also add edit controls to a custom dialog, which allow the end user to enter a single line of text. InstallShield has standard dialogs called **SdShowDlgEdit1**, **SdShowDlgEdit2**, and **SdShowDlgEdit3**, which display one, two, or three edit fields in which the end user can enter text.

You can read the text stored in an Edit control with **CtrlGetText**. For example, to read the text from an Edit control with resource ID 10000 into a string variable called svEdit, you can use the following code:

```
CtrlGetText("CustomDialog", 10000, svEdit);
```

Similarly, you can set the initial text stored in an Edit control (in the DLG_INIT block of the message-loop's switch statement) with **CtrlSetText**.



Tip *• You can use the* ***Password*** *attribute of an edit control to hide the characters that the end user types into the edit field. You can use the* ***Number*** *attribute to allow the end user to enter only numbers in the edit field.*

Using List Box and Combo Box Controls

List box and combo box controls need to be associated with a variable of type LIST. Before displaying the list box control, you should populate a string-list variable, and associate it with the list box or combo box control using **CtrlSetList** in the dialog's DLG_INIT handler. (For a combo box control, you will usually set the Drop-Down List property to True, and set the Height property to the height of the drop-down-list portion of the control.)

To set the initial selection in a list box or combo box control, you can call **CtrlSetCurSel** in the dialog's DLG_INIT handler; and to retrieve the user's current selection call **CtrlGetCurSel**. For example, the following code populates a list variable with the drive letters of every available drive (using GetValidDrivesList), associates the list with a combo-box control, and then reads the end user's selection from the combo box before exiting the dialog.

```
function NUMBER CustomDialog( )
  NUMBER nReturn;
  NUMBER nControl;
BOOL bDone;
       // variables for combo box list and current selection
  LIST listDrives;
  STRING svDrive;
begin
    nReturn = EzDefineDialog("CustomDialog", ISUSER, "CustomDialog", 0);

    bDone = FALSE;

    // create the list containing the combo box items
    listDrives = ListCreate(STRINGLIST);
    // fill the list with all available drive letters
    GetValidDrivesList(listDrives, -1, -1);

    while (!bDone)

    nControl = WaitOnDialog("CustomDialog");

    switch (nControl)

        case DLG_INIT:
            // Initialize the back, next, and cancel button enable/disable states
            // for this dialog and replace %P, %VS, %VI with IFX_PRODUCT_DISPLAY_NAME,
            // IFX_PRODUCT_DISPLAY_VERSION, and IFX_INSTALLED_DISPLAY_VERSION,
            // respectively, on control IDs 700-724 and 202.
            hwndDlg = CmdGetHwndDlg("CustomDialog");
            SdGeneralInit("CustomDialog", hwndDlg, 0, "");
            CtrlSetState("CustomDialog", MYCHECKBOX1, BUTTON_CHECKED);
            // associate the list with the combo box
            CtrlSetList("CustomDialog", MYCOMBOBOX1, listDrives);
            // get the first drive letter from the list...
            ListGetFirstString(listDrives, svDrive);
            // ...and make it the current selection
            CtrlSetCurSel("CustomDialog", MYCOMBOBOX1, svDrive);
        // ...cases for other controls...

    endswitch;
```

```
    endwhile;

    // get the end user's selection, and display it in a message box
    CtrlGetCurSel("CustomDialog", MYCOMBOBOX1, svDrive);
    MessageBox("You selected drive " + svDrive, INFORMATION);

    EndDialog("CustomDialog");
    ReleaseDialog("CustomDialog");

    return nReturn;
end;
```

List box and combo box controls have a Sorted property, which you can set to Yes to sort the contents of the list controls by the items' display names.

Editing Dialog Behavior in an InstallScript or InstallScript MSI Project



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

To change the behavior of a dialog, you can modify the dialog function's parameters according to how you want the dialog to behave. To learn about the available parameters, refer to the dialog function documentation:

- Dialog Functions
- Dialog Customization Functions

Adding a Field That Contains a Product Name, Product Version, or Installed Version in Sd Dialog Static Text Fields



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

InstallShield enables you to place your product name, product version, or installed version globally in Sd dialog static text fields containing the placeholder %P, %VS, or %VI (sometimes appearing as an extra space). At run time, the values of the system variables IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and IFX_INSTALLED_DISPLAY_VERSION are displayed in place of %P, %VS, and %VI in the static text fields.

If you are adding a static text field containing a placeholder, give the static text field a unique (to that dialog) ID in the range of 701 through 799. IDs in the 701 through 799 range instruct InstallShield to scan the static text field for the existence of the placeholders. If a placeholder is found, InstallShield replaces it with the value of the corresponding system variable.

Using an HTML Control on a Dialog

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

InstallShield includes support for HTML controls on dialogs in InstallScript and InstallScript MSI projects. HTML controls enable you to use HTML markup for dialog controls. You can include on dialogs links to Web pages, installed HTML files, and HTML support files. If an end user clicks the hyperlink on the run-time dialog, you can have the HTML page open in an Internet browser, or you can trigger other behavior that you have defined through your InstallScript code. The HTML control lets you use any valid HTML markup, including styles to control their appearance.

The HTML control also lets you display the HTML content directly on a dialog if the content is an installed HTML file or HTML support file.

To create an HTML control, you need to start with a static text control. Once you have placed the static text control on a dialog, you can convert it to be an HTML control.

Task ***To convert any static text control on a dialog to be an HTML control, do one of the following:***

- In the Dialogs view, set the Text property of the control as follows:

  [html] *HTML Markup*

- Use InstallScript to set the text of a control to be **[html]**, followed by the HTML markup.

At run time, the InstallScript engine converts the control to an HTML control.

Example

The following example demonstrates how to convert a static text control to an HTML control. The sample HTML code enables you to set the colors and fonts of the control so that they match those of the dialog.

Task ***To convert a static text control to an HTML control by using the control's Text property:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click the dialog that you want to contain the HTML control, and then click **Edit**. InstallShield displays the dialog in the Dialog Editor, using the project's default language.
3. Click the static text control that you want to convert to an HTML control. InstallShield displays its properties in the right pane.
4. In the **Text** property, click the ellipsis button (**...**). InstallShield displays a list of all of the current strings in the project.
5. Click the **Edit** button.
6. Set the value of the string identifier as follows:

```
[html]<style type="text/css">html,body {padding:0; margin:0; background-color:ButtonFace} * {font-
size: 8pt; font-family: "MS Sans Serif";} </style> <a href="http://www.MyWebSite.com">Visit my Web
site</a>
```

At run time, the installation uses the default InstallScript dialog font (8 point MS Sans Serif) and sets the background color to the Windows dialog color, which is usually gray. If the content is on a dialog area that is white, you can omit the *background-color:ButtonFace* style part so that the background of the HTML control becomes white.



Tip *• As an alternative to converting the static control through the Dialogs view, you could edit the dialog's script function in the InstallScript view. For example, add the following code to the DLG_INIT case in the dialog message loop:*

```
CtrlSetText( szDlg, HTML_CTRL_ID, "[html]<style type=\"text/css\">html,body {padding:0; margin:0;
background-color:ButtonFace} * {font-size: 8pt; font-family: "MS Sans Serif";} </style><a href=\"http:/
/www.MyWebSite.com\">Visit my Web site</a>" );
```

For sample script, see CtrlGetUrlForLinkClicked Example.

Handling Click Events for Links in HTML Controls

If you use the **CtrlSetText** function to set HTML control content at run time and the control creation succeeds, **CtrlSetText** returns 0. If it fails, **CtrlSetText** returns ISERR_GEN_FAILURE.

To handle click events for links in HTML controls, the **WaitOnDialog** function returns the control ID of the HTML control (which is also the control ID of the original static text control). A custom dialog script function can contain a case statement that handles the HTML control ID, and then calls the **CtrlGetUrlForLinkClicked** function to obtain the URL for the link that was clicked. With this link, the script can take any action (such as launching an Internet browser to navigate to the link).

The following behavior occurs when an end user clicks a link for an HTML control that uses the anchor tag:

- If the anchor tag does not include the target attribute, or if the target value does not result in a new window, the control ID is returned to the dialog script. This allows for the script to handle link clicks itself.
- If the anchor tag does include the target attribute and its value is _blank or some other equivalent value that results in a new window, no message is sent to the script, and the URL is automatically launched.

Special Considerations for HTML Controls

If you are adding HTML controls to dialogs, note the following details.

Run-Time Requirements for HTML Controls

The HTML control requires Internet Explorer 5.5 or later for full functionality. If an earlier version is present on a target system, the only way to set the content for the HTML control is to provide an HTML file to **CtrlSetText**. Note that Internet Explorer does not need to be the default browser on the target system.

HTML Control Support for HTML Files

If you using an HTML file with an HTML control, you must specify the full path to the file through the **CtrlSetText** function. Otherwise, the InstallScript engine will not know where the file is located. Following is sample InstallScript:

```
CtrlSetText(szDlg, HTML_CTRL_ID, "[html]file://" + SUPPORTDIR^"aboutpage.htm");
```

Using an HTML Control for an Email Address

If you want to add a hyperlink that points to an email address, you can use code such as the following:

```
CtrlSetText(szDlg, HTML_CTRL_ID, "[html]<a
href=\"mailto:support@mycompany.com?Subject=Example%20Line\">Send email to Support</a>");
```

When an end user clicks such a link, a new email message is automatically opened in the target system's email application. Note that with the mailto link, no message is sent to the script.

HTML Text Content Is Not Formatted

Any content set in an HTML control is passed as is. No formatting of the content is performed. If you need to format any of the content, you can manipulate the content string in script and then pass the content string to **CtrlSetText**.

Character Count Limits for the Text Property of Static Text Controls

The Text property of static text controls has a maximum limit of 256 characters. Therefore, if you convert a static text control to an HTML control by adding **[html]** to the beginning of the text content for a control, the maximum number of characters that you can enter is 256; this includes the hyperlinked text as well as all characters for any HTML markup that you specify. If you need to enter more than 256 characters for the content, use the **CtrlSetText** function to create the control.

Reverting to the Default Dialog



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

If you have edited a dialog and you later decide that you want to undo all of your changes, you can revert back to the original default dialog.



Task ***To revert to the default dialog:***

1. In the View List under **User Interface**, click **Dialogs**.
2. Right-click the edited dialog and click **Revert**.

Resource Compiler and Resource Linker



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

To modify dialogs in InstallScript or InstallScript MSI projects, you must have a resource compiler and resource linker installed on your system. InstallShield includes both of these types of tools.

Task ***To view the location of the compiler and linker:***

1. On the **Tools** menu, click **Options**. The **Options** dialog box opens.
2. Click the **Resource** tab.

The Resource tab lists the locations of the compiler and linker.

Task ***To replace the resource compiler or resource linker with one that is already on your system:***

1. On the **Tools** menu, click **Options**. The **Options** dialog box opens.
2. Click the **Resources** tab.
3. Click the Browse button next to the **Resource compiler location** box or the **Resource linker location** box to navigate to the program file.

Dialog Sampler

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The Dialog Sampler is an InstallShield wizard that displays samples of all built-in dialogs as well as dialogs called by script dialog (sd) functions in InstallScript and InstallScript MSI projects. You can launch two different varieties of the Dialog Sampler.

Task ***To launch the Dialog Sampler:***

On the **Tools** menu, point to **InstallScript**, and then click either **Standard Dialog Sampler** or **Skinned Dialog Sampler**.

You can also launch these samplers using the shortcut in the InstallShield program folder on the Programs menu.

Dialog Skins

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Overview

Dialog skins enable you to display end-user dialogs with a different look and color scheme. Skinned dialogs are slightly larger than standard dialogs, and the controls are located differently. For skinned dialogs to display properly, the desktop area on the target machine must be at least 800 by 600 pixels (or if large fonts are used, 1024 by 768 pixels), and the system must display at least 16-bit color.

Limitations

There are limitations when using dialog skins in your project:

- A dialog skin cannot be applied to or removed from a standard dialog once you have edited that dialog in the Dialog Editor. Therefore, if you want to specify skins for dialogs that you want to modify, specify the skin first, and then edit it.
- The location of the standard navigation buttons (Next, Cancel, Back, Finish) are the same for all skins. The .skin file controls their position. Moving them in the Dialog Editor has no effect on the run-time positioning. There are also some Browse buttons on a few dialogs (for example, **SdAskDestPath**) that cannot be repositioned.
- Skinned dialogs cannot use windowed billboards. To learn more about the windowed style of billboard, see Billboard Styles and File Types for InstallScript and InstallScript MSI Projects.
- If you add custom dialogs to your project, the new dialogs should be the same size as the other end-user dialogs. If the custom dialogs are a different size, they may not be displayed correctly; for example, the navigation buttons might not be visible to the end user during run time, due to the positioning issue described above.

Note *• If you want to specify a dialog skin for a custom dialog, you must set the skin before creating the custom dialog in the Dialog Editor in order for the skin to appear properly.*

Specifying Dialog Skins

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

You can use a dialog skin to change the look of the end-user dialogs in your installation. You can specify one skin per installation project. All of the dialogs in your project are displayed using the selected skin.

Note *• A dialog skin cannot be applied to or removed from a standard dialog once you have edited that dialog in the Dialog Editor. Therefore, if you want to specify skins for dialogs that you want to modify, specify the skin first, and then edit it.*

If you want to specify a dialog skin for a custom dialog, you must set the skin before creating the custom dialog in the Dialog Editor in order for the skin to appear properly.

Selecting a Dialog Skin

Task ***To select a dialog skin:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **Skins** item to see the available dialog skin options.
3. Click a skin to see a preview dialog displayed in the right pane.
4. To select a displayed skin, click **Select** in the preview pane. A red check mark appears on the selected skin's icon in the **Dialogs** explorer.

All of the dialog skins use .gif files for the fade graphic, with the exception of the Blue skin. The Blue skin uses a .bmp file, which results in a larger file size. The Blue skin's appearance on a 16-bit color system is not as clean as the other colors when a .gif file is used. If you want to use a Blue skin that supports 16-bit color systems and file size is not an issue, select the Blue option. For a smaller file size, select the BlueTC (True Color) option, which uses a .gif file for the graphic.

Clearing Dialog Skin Selections

Task ***To clear the dialog skin selection:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **Skins** item to see the available dialog skin options, and select the **<None>** item.
3. Click **Select** in the preview panel.

Working with Dialogs in Basic MSI Projects

The information in this section of the documentation explains how to create and perform other basic functions with dialogs in Basic MSI projects.

Task ***The process of creating a new dialog and displaying it in your installation can be broken down into the following tasks:***

1. Add the dialog to the project.
2. Edit the dialog's layout.
3. Define the controls' behavior (under what conditions they should be displayed, the events that their interaction should trigger, and the events that they should subscribe to).
4. Display the dialog with a NewDialog or SpawnDialog event in another dialog's control or by inserting the dialog into your project's sequences.

Displaying Dialogs During Basic MSI Installations

You can display an end-user dialog in one of two ways, as described below:

- Insert the dialog into a User Interface sequence.
- Launch the dialog through another dialog's behavior.

Inserting Dialogs into Sequences



Project • *When you create a dialog in a merge module, you cannot insert it into a sequence until you associate that module with an installation project.*



Task ***To schedule a dialog in a sequence:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. In the **Sequences** explorer, expand one of the two high-level sequences in which dialogs are usually displayed—**Installation**, **Advertisement**, or **Administration**—to view its **User Interface** sequence.
3. Expand the **User Interface** sequence to see how the existing actions, dialogs, and custom actions are ordered.
4. Right-click the existing action or dialog that you want your dialog to follow in the sequence, and then click **Insert**. The Insert Action dialog box opens.
5. In the list, select **Dialogs**.
6. Select your new dialog from the list of dialogs.
7. Click **OK**.

When your installation runs with a full user interface (determined by the `/q` command-line parameter) and all conditions are met for this dialog, it will be displayed within the selected sequences.

While InstallShield lets you insert a dialog into an Execute sequence, doing so violates ICE13.

Launching a Dialog Through Another Dialog's Behavior

Most dialogs are displayed as a result of the NewDialog (displays another dialog in the wizard) or SpawnDialog (shows a modal dialog) control event. By selecting this event in a dialog's behavior, usually for a Next button, you can specify the dialog that succeeds it. By adding a condition to the control event, you could present different dialogs depending on the result of a selection in the preceding dialog.



Task ***For example, to display a WelcomeBitmap dialog after the InstallWelcome dialog:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **InstallWelcome** dialog, and then click its **Behavior** subnode.
3. In the pane that contains a list of controls and the corresponding control types, select the **Next** button.
4. In the **Events** setting, click the **New Event** button, and then click **NewDialog**. InstallShield adds some **NewDialog** subsettings under the **Event** setting.

5. In the **NewDialog** setting, enter the following: **1**

   A condition of **1** indicates that the dialog should be created under all conditions.

6. In the **Dialog Name** setting, enter the following: **WelcomeBitmap**



***Note** • If the InstallWelcome dialog had launched a different dialog from the Next button, you would repeat the above procedure for the WelcomeBitmap's Next button to show the next dialog.*

To learn about viewing your new dialog in the Custom Actions and Sequences view, see Viewing End-User Dialog Order in the Custom Actions and Sequences view (Basic MSI Projects).

Creating New Dialogs in Basic MSI Projects



Task ***To create a new dialog in a Basic MSI project:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, right-click **All Dialogs** and click **New Dialog**. The Dialog Wizard launches to help you create a new dialog.

The next step is to edit the dialog's layout and the controls' behavior.



***Tip** • You can also add a dialog by importing one from another project.*

Displaying a Dialog in the End-User Interface

Adding a dialog to a project does not mean that the dialog will be displayed in the installation. For information on inserting the dialog into one or more of the project's sequences, see Displaying Dialogs During Basic MSI Installations.

Editing Dialog Layout in Basic MSI Projects

Using the Dialog Editor, you can modify a dialog, edit its controls, and set display properties for the dialog and its controls, all in a visual environment.



***Note** • When an end user runs your installation on a Windows platform that uses a desktop display theme, the theme is used to display your installation's end-user dialogs.*

The Dialogs view manages versions of the dialog for each of your project's supported languages. You must select a language-specific version in order to edit the layout. These versions remain identical except for changes you make to a control's size. For more information, see Modifying Dialogs for Each Language.

Opening the Dialog Editor

Task ***To open a dialog in the Dialog Editor, do any of the following:***

- Select the language-specific version of the dialog under the dialog's name in the **Dialogs** view.
- In the **Custom Actions and Sequences** view, right-click the dialog and click **Edit Layout**.

The Dialog Editor opens in the right pane of the Dialogs view.

Adding a Control to a Dialog in a Basic MSI Project

You can use the Controls toolbar in InstallShield to add controls to a dialog.

Task ***To add a control to a dialog:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **All Dialogs** folder, and then expand the dialog item that should have the new dialog control.
3. Click a language under the dialog item that you expanded. The Dialog Editor in the center pane shows the dialog in the selected language.
4. In the **Controls** toolbar, click the control that you want to add.
5. Draw a rectangle on the dialog in the location where you want the control to be displayed.
6. In the right pane, set the control's properties.

Tip • *If you are editing an InstallShield project from within Microsoft Visual Studio, you can use the Toolbox to add controls.*

Configuring the Controls' Settings

The dialog itself and every control on it has a grid of settings in the Dialog Editor. These settings determine characteristics such as a control's size and position, its default text (as a string identifier), or whether the dialog is modal.

To edit the settings of a control, select the control on the dialog. The settings vary depending on the type of control:

- Bitmap
- Dialog
- Check Box
- Push Button
- Edit Field
- Combo Box

- Text Area
- List Box
- Radio Button
- Bitmap
- Group Box
- Billboard
- Line
- Radio Button Group
- Selection Tree
- Progress Bar
- List View
- Scrollable Text
- Icon
- Directory List
- Directory Combo
- Volume Cost List
- Volume Select Combo
- Masked Edit
- Path Edit

Displaying Controls on Top of a Bitmap



Task ***To display controls on top of a bitmap:***

- The value for the **Tab Index** property of the bitmap must be lower than the value for the **Tab Index** property of any controls that are to be placed on top of the bitmap. The easiest way to configure this is to set the bitmap control's **Tab Index** property to 0.
- All controls—including the bitmap—must have their **Tab Stop** properties set to **True**.

Editing Dialog Behavior in Basic MSI Projects

In the Dialogs view, you can configure settings to specify how controls will behave at run time. The behavior-related settings for a control enable you to configure the following types of functionality:

- Events that an end user interacting with the control will trigger
- Windows Installer events to which a control needs to subscribe
- Conditions under which the control will be displayed

Task ***To access the behavior settings for a control:***

1. Do one of the following:
   - In the **Dialogs** view, expand the dialog node for the dialog that you want to modify, and then click its **Behavior** subnode.
   - In the **Custom Actions and Sequences** view, right-click the dialog and click **Edit Behavior**.
2. In the center pane, which shows a list of all of the controls on the selected dialog, click the control that you want to configure. Its settings are displayed in the right pane.
3. Configure the **Events**, **Subscriptions**, and **Conditions** settings as needed.

Using Control Events in Basic MSI Dialogs

Control events enable you to provide custom functionality for each control on your dialogs. With control events, you can launch custom actions, spawn new dialogs, or display a progress bar. Many control events make use of published information. In some cases, your controls must subscribe to control-event information in order to make use of it.

To associate an event with a control, use the Behavior area for a dialog in the Dialogs view. For more information, see Editing Dialog Behavior in Basic MSI Projects. To learn about subscriptions for controls, see Using Subscriptions in Basic MSI Dialogs.

To specify a condition for any event, click the ellipsis button (...) in the event setting.

Following is a list of the available control events:

Table 4-3 • Types of Events for Controls

Control Event	Description
AddLocal	Sets the specified feature (or all features, if **ALL** is specified) to be installed locally. To specify the feature, configure this setting's subsetting.
AddSource	Sets the specified feature (or all features, if **ALL** is specified) to be installed to run from source. To specify the feature, configure this setting's subsetting.
CheckExistingTargetPath	Verifies that the specified path can be written to, and prevents additional control events from being triggered unless the specified path exists. To specify the property that contains the path, configure this setting's subsetting.
CheckTargetPath	Prevents additional control events from being triggered unless the specified value is a valid path. To specify the property that contains the path, configure this setting's subsetting.
DirectoryListNew	Notifies the directory list control that a new folder must be created. The installer then creates a new folder and allows the user to enter a new name for the folder.

Table 4-3 • Types of Events for Controls (cont.)

Control Event	Description
DirectoryListOpen	Notifies the directory list control to open the selected directory. If none is selected, it prevents additional control events from being triggered.
DirectoryListUp	Notifies the directory list control to go up a level, selecting the parent of the current directory.
DoAction	Triggers a custom action when the selected control is activated. To specify the custom action, configure this setting's subsetting.
EnableRollback	Enables or disables rollback functionality for the installation. To specify whether you want the event to enable rollback, configure this setting's subsetting. A value of True enables rollback; False disables it.
EndDialog	Dismisses a modal dialog. You can use this event for a push button control or a selection tree control. To specify the behavior that should occur for this event, configure this setting's subsetting. For regular dialogs, available options are: • **Exit**—The dialog sequence closes and control is returned to the installer with the UserExit value. This option cannot be used with child windows. • **Retry**—The dialog sequence closes and control is returned to the installer with the Suspend value. This option cannot be used with child windows. • **Ignore**—The dialog sequence closes and control is returned to the installer with the Finished value. This option cannot be used with child windows. • **Return**—The dialog exits and control is returned to the parent window. If no parent exists, control is returned to the installer with the Success value. For error dialogs, available options are: • ErrorOK • ErrorCancel • ErrorAbort • ErrorRetry • ErrorIgnore • ErrorYes • ErrorNo

Table 4-3 • Types of Events for Controls (cont.)

Control Event	Description
MsiLaunchApp	 ***Note*** • *Windows Installer 5 includes support for this event. Earlier versions of Windows Installer ignore this event. Therefore, if your installation is run on a system that has Windows Installer 4.5 or earlier and you want to use this event, add a condition to the dialog control so that it is not displayed on systems that have Windows Installer 4.5 or earlier.* Runs a specified file. This event is typically used for a check box control on the SetupCompleteSuccess dialog. The check box enables end users to choose whether to run the application at the end of the installation. The check box control should include a condition that prevents the control from being displayed during uninstallation. To specify the command line for launching the file, configure this setting's subsetting.
MsiPrint	 ***Note*** • *Windows Installer 5 includes support for this event. Earlier versions of Windows Installer ignore this event. Therefore, if your installation is run on a system that has Windows Installer 4.5 or earlier and you want to use this event, add a condition to the dialog control so that it is not displayed on systems that have Windows Installer 4.5 or earlier.* Enables end users to print the contents of the control. You can add this event to a push button control that is on a dialog that has a scrollable text control. When an end user clicks the push button control, the contents of the scrollable text control are printed.
NewDialog	Advances to another dialog. To specify the new dialog, configure this setting's subsetting.
Reinstall	Reinstalls the specified feature (or all features, if **ALL** is specified). To specify the feature, configure this setting's subsetting.
Remove	Indicates that the specified feature (or all features, if **ALL** is specified) is selected for removal. To specify the feature, configure this setting's subsetting.
Reset	Resets all of the property changes that all of the controls on the dialog have performed to their original values.
RMShutdownAndRestart	Indicates that the Restart Manager should shut down all applications that have files in use and restart them at the end of the installation.

Table 4-3 • Types of Events for Controls (cont.)

Control Event	Description
SelectionBrowse	Launches the specified Browse dialog that enables the end user to modify the path of the highlighted item. To specify the dialog, configure this setting's subsetting.
SetInstallLevel	Sets the **INSTALLLEVEL** property to the specified value. To specify the appropriate installation level, configure this setting's subsetting.
SetProperty	Sets the value of a property. To specify the property and value, configure this setting's subsettings.
SetTargetPath	Checks and sets the selected path. To specify the property that contains the path, configure this setting's subsetting.
SpawnDialog	Launches the specified dialog of a modal dialog without closing the current dialog.
SpawnWaitDialog	Launches a dialog that waits until the conditional expression evaluates as True; then the dialog closes.
ValidateProductID	Sets the ProductID property.

Using Subscriptions in Basic MSI Dialogs

The Windows Installer service provides volumes of information to the installation during the run time. One of the most useful ways to gather information during installation is to use subscriptions. To subscribe to something is commonly defined as entering one's name for a publication or service. In Windows Installer terms, actions generally publish information, and dialog controls generally subscribe to information.

Examples of Common Uses for Subscriptions

The best example of this relationship centers around progress bars. The InstallFiles action publishes information on the percentage of files moved and the percentage remaining to be moved. When a progress bar subscribes to this information, it is able to accurately display the progress of the installation's file transfer process.

Windows Installer tracks progress through what are called *ticks*. When your progress bar subscribes to this information, it is passed the **ticksSoFar** and **totalTicks** values. The progress bar uses this information to display the total progress of the installation.

Subscriptions are not used only for progress bars. You might have a custom action that validates a serial number as part of your installation. This action could publish the failure or success of validation. On a dialog, you could have a Next button that subscribes to this information. If the action publishes the failure of the validation, the button remains disabled. If the action publishes success, the button is enabled.

Types of Subscriptions

To set up a subscription for a control, use the Behavior area for a dialog in the Dialogs view. For more information, see Editing Dialog Behavior in Basic MSI Projects.

Following is a list of the available events to which controls can subscribe:

Table 4-4 • Types of Events to Which Controls Can Subscribe

Event	Description
ActionData	Displays information about the latest action. A text control can subscribe to this event to display information such as the names of files that are being copied during the installation.
ActionProgress	Displays progress information about an action that is being monitored. A progress bar control can subscribe to this event. To specify the action, configure this setting's subsetting.
ActionText	Displays the name of the present action. A text control can subscribe to this event to display a description of the current action that is being performed during the installation.
IgnoreChange	Highlights a folder in the current directory without opening the folder.
ScriptInProgress	Displays informational text while Windows Installer is compiling the installation's execution script.
SelectionAction	Displays a UI string that describes the highlighted item. A text control can subscribe this event to display the UI string.
SelectionDescription	Displays a UI string in a text control. The UI string describes the feature that is selected in the selection tree control.
SelectionNoItems	Removes description text or disables an unnecessary button.
SelectionPath	Publishes the path of the item that is selected in the selection tree control. A text control can subscribe this event to display the path.
SelectionPathOn	Publishes a Boolean value that indicates whether a selection path is associated with the currently selected feature.
SelectionSize	Publishes the size of the highlighted item.
SetProgress	Publishes information about the installation's progress.
TimeRemaining	Publishes the approximate number of seconds that remain for the current progress sequence. A text control can subscribe this event to display the remaining time.

Triggering Control Events in Basic MSI Dialogs

An event is a predefined action that occurs when an end user interacts with a control. Events are defined in the Behavior area of the Dialogs view. For more information on each of the events that are available, see Using Control Events in Basic MSI Dialogs.

To view a control's events, select it in the list of control names and then configure the Events setting in the right pane.

Adding New Events to a Dialog Control



Task ***To add a new event to a dialog control:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the dialog node for the dialog that you want to modify, and then click its **Behavior** subnode.
3. In the center pane, which shows a list of all of the controls on the selected dialog, click the control that you want to configure. Its settings are displayed in the right pane.
4. In the **Events** setting, click the **New Event** button, and then select the type of event that you want to add. InstallShield adds a new set of rows under the **Events** setting.
5. To specify a condition for the event, click the ellipsis button (**...**) in the new event setting. The **Condition Builder** dialog box opens, enabling you to specify the conditions that must be met for the action.

   If you want the event to be triggered under any conditions, enter the following condition:

   1
6. If the event setting has a subsetting, configure the subsetting as needed.

Reordering a Dialog Control's Events

The events that are configured for a user interface control are launched at run time in the order in which they are displayed for that control's Events setting.



Task ***To move an event:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the dialog node for the dialog that you want to modify, and then click its **Behavior** subnode.
3. In the center pane, which shows a list of all of the controls on the selected dialog, click the control that has the events that you want to reorder. Its settings are displayed in the right pane.
4. In the **Events** area of the grid of settings, find the event setting of an event that you want to move.
5. In the event setting for an event that you want to move, click the **Move Event** button, and then click **Move Up** or **Move Down**.

Removing an Event from a Dialog Control

Task ***To remove an event from a control:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the dialog node for the dialog that you want to modify, and then click its **Behavior** subnode.
3. In the center pane, which shows a list of all of the controls on the selected dialog, click the control that has the event that you want to remove. Its settings are displayed in the right pane.
4. In the **Events** area of the grid of settings, find the event setting of the event that you want to remove.
5. In the event setting, click the **Delete Event** button.

Launching Custom Actions from Dialog Buttons

To launch a custom action from a button on a dialog, use the button control's DoAction event. Before you can set up a DoAction event, you need to create a custom action.

When you have created a custom action, you can set up a DoAction event to launch this action.

Task ***To launch a custom action from a push button control:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the dialog node for the dialog that you want to modify, and then click its **Behavior** subnode.
3. In the center pane, which shows a list of all of the controls on the selected dialog, click the push button control that you want to trigger the custom action. Its settings are displayed in the right pane.
4. In the **Events** setting, click the **New Event** button, and then click **DoAction**. InstallShield adds a new set of rows under the **Events** setting.
5. To specify a condition for the event: In the **DoAction** setting, click the ellipsis button (**...**). The **Condition Builder** dialog box opens, enabling you to specify the conditions that must be met for the action.
6. In the **Action** subsetting, specify the name of the action that you want to associate with the control.

Build and test your installation to ensure that it works as expected.

Viewing End-User Dialog Order in the Custom Actions and Sequences view (Basic MSI Projects)

Project • *This information applies to Basic MSI projects. To view the order of your end-user dialogs in an InstallScript project or an InstallScript MSI project, use the InstallScript view to review your script.*

In the Custom Actions and Sequences view, you can view primary and next dialogs in the order in which they appear to the end user during the run time of your installation. Primary dialogs are top-level dialogs that have been inserted into the installation sequence. You can change the order of a primary dialog within the Custom Actions and Sequences view.

Next dialogs are dialogs that are triggered by the action of a previous dialog's Next button control events. Although you can view next dialogs in the Custom Actions and Sequences view, they are not a part of the installation sequence, and you cannot change their order in the Custom Actions and Sequences view. To change the order of a next dialog, you must use the Behavior node under a dialog in the Dialogs view. To learn more, see Editing Dialog Behavior in Basic MSI Projects.

Initial Sequences Explorer

When you first open the Custom Actions and Sequences view, the Sequences explorer lists all primary dialogs and actions in the order in which they will be executed during your installation. Many of the next dialogs have plus signs (+) next to them.

Expanded Sequences Explorer

In addition to viewing the order of primary dialogs that have been inserted into the installation sequence, you can also view next dialog order in the Sequences explorer.

When you click the plus sign (+) next to a primary dialog, the list expands to display the order in which the primary dialog's next dialogs—if any—will appear during the run time. If a particular dialog has more than one Next button control event, all possible next dialogs are displayed under the dialog.

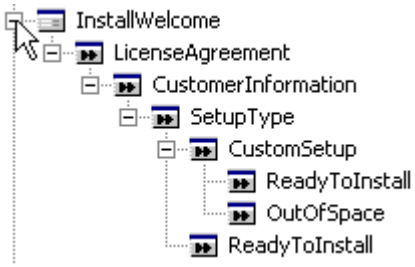


Figure 4-3: Viewing the Order of Dialogs in the Sequence

Next Dialog Information Panel

When you click a next dialog, the right pane displays information about the dialog. The right pane also has hyperlinks to the behavior and layout editors, as well as help information. More detailed information about the selected dialog is available in the Dialogs view.

Context Menu Commands

When you right-click a primary dialog, a context menu displays commands for you to edit the sequence, refresh the dialog sequence view, or edit the dialog's layout or behavior. When you right-click a next dialog, a context menu displays commands for you to edit the dialog's layout or behavior.

Edit Behavior



Task ***To move to the Dialog Behavior Editor from within the Sequences explorer in the Custom Actions and Sequences view, do either of the following:***

- Right-click the next dialog that you want to edit and click **Edit Behavior**.
- Click the name of the dialog that you want to edit. Then, in the **Action Items** section of the right pane, click **Edit dialog behavior**.

For more information, see Editing Dialog Behavior in Basic MSI Projects.

Edit Layout



Task ***To move to the Dialog Layout Editor from within the Sequences explorer in the Custom Actions and Sequences view, do either of the following:***

- Right-click the next dialog that you want to edit and click **Edit Layout**.
- Click the name of the dialog that you want to edit. Then, in the **Action Items** section of the right pane, click **Edit dialog layout**.If you have written your installation in more than one language, you must select the appropriate language in the list.

For more information, see Editing Dialog Layout in Basic MSI Projects.

CustomSetup Dialog Options



Project • *This information applies to the following project types:*

- *Basic MSI*
- *MSI Database*
- *Transform*

The CustomSetup dialog has a sophisticated end-user interface that is tightly integrated with information about the target system, the features in your installation, and Windows Installer installation options. It provides the end user with the most control over the installation.

Many of the options and information it offers are determined by the feature properties that you set in your setup design, as described below.

Advertisement Option

Feature advertisement enables files to be installed on demand after the installation has initially run. In the CustomSetup dialog, when end users click a feature, they can specify that they want it installed later by selecting the **Will be installed when required** option.

However, that default option is present only if you select Allow Advertise or Favor Advertise for the feature's Advertised setting. To prevent Windows Installer from displaying the option to advertise the feature, select Disallow Advertise for this setting.

Hiding Features

When you set a feature's Display setting to Not Visible, the end user cannot see the feature or its subfeatures in the CustomSetup dialog and therefore cannot change any of its installation options.

Displaying All Subfeatures

The feature's Display setting also governs whether its subfeatures are expanded when the dialog first appears with the following options:

Table 4-5 • Options for the Display Setting of a Feature

Option	Description
Visible and Collapsed	The feature is displayed in the CustomSetup dialog with its subfeatures collapsed by default.
Visible and Expanded	The feature is displayed in the CustomSetup dialog with its subfeatures expanded by default.

Setting Default Destination Folders for Features

An end user can view and edit the folder to which a feature will be installed by selecting the feature in the CustomSetup dialog and clicking the Details button. The feature's Destination setting is the default value that appears, and it is reassigned to a new value if the end user enters or browses to a new path in the resulting Feature Details dialog.

For a complete discussion of the factors that affect where a feature is installed, see Setting a Feature's Destination.

Naming Features

Specify a different name to appear in the CustomSetup dialog by setting its Display Name.

Displaying Feature Descriptions

The description that is shown at the bottom of the CustomSetup dialog when you select a feature is taken from its Description setting.

Changing the Feature Order

You can change the order in which the features appear to the end user in the CustomSetup dialog. The order in which features are displayed is dictated by the order in the Setup Design view. For more information, see Reordering Features.

Requiring Features To Be Installed

If you set a feature's Required setting to Yes, the end user does not see the **Will not be available** option, meaning that the feature must be installed.

Dialog Themes

Project *• Dialog themes are available in Basic MSI projects.*

Dialog themes are predefined sets of images that give your end-user dialogs a unified and distinctive look.

With the click of a button, you can select one of the available themes for your project, and InstallShield applies that theme to all of the interior and exterior dialogs, as well as the **Setup.exe** initialization dialog, in your project. You can easily preview each dialog from within the Dialogs view to see how it looks with the selected theme.

Note *• InstallShield does not currently let you create your own dialog themes. However, InstallShield includes many themes. For more information, see Available Themes and Corresponding Dialog Sizes.*

Run-Time Requirements for the Wide Dialog Themes

Project *• Dialog themes are available in Basic MSI projects.*

If you use one of the wide dialog themes, the screen resolution on target systems must be a minimum of 800×600 pixels (or, if large fonts are used, 1024×768 pixels). If the resolution is lower than that, a horizontal scrollbar is added at run time to the bottom of each of the dialogs that is displayed with the wide theme.

Previewing a Dialog Theme

Project *• Dialog themes are available in Basic MSI projects.*

When you select a different theme for your project, InstallShield applies that theme to all of the interior and exterior dialogs in the Dialogs view. You do not need to build and run your installation in order to view the dialogs.

Task ***To preview a theme before selecting it for your project:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **Themes** folder.
3. Click one of the themes under the **Themes** folder.

In the right pane, InstallShield shows a screen shot of a sample exterior dialog with the selected theme.

Task ***To preview how a dialog in your project looks with the selected theme:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **All Dialogs** folder, and then expand the dialog item that you want to preview.

3. Click a language under the dialog item that you expanded.

In the center pane, InstallShield displays a preview of the dialog with the selected theme applied.

Selecting or Changing a Dialog Theme



***Project** • Dialog themes are available in Basic MSI projects.*

You can use a dialog theme to change the look of the end-user dialogs in your installation. You can select one theme per project.



***Tip** • If you switch from one of the wide-style themes to a standard-width theme, the positions of any controls that are placed along the far left or far right sides of the dialog may change.*

If you want to change the theme for your project and you also want to add a custom exterior dialog, change the theme first, and then add the custom exterior dialog. Otherwise, the dialog may not appear properly.



Task ***To change the dialog theme used for a project:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **Themes** folder.
3. Right-click the theme that you want to use, and then click **Select**.

InstallShield applies the selected theme to the dialogs in your project. In addition, InstallShield displays a red check mark on the selected theme's icon in the Dialogs explorer.



***Tip** • You can also change the theme by clicking a button: In the Dialogs explorer, click the theme that you want to use. Then, in the right pane, click the Select button.*

Background Information on How InstallShield Applies Themes

If you switch from one dialog theme to another, InstallShield applies the theme to any dialogs in your project that match a specific width and height. Therefore, if you change the width or height of any of the dialogs in your project, and then you change the theme, InstallShield does not apply the theme to any dialogs that have a custom width or height.

For example, if you switch from a standard width theme to any other theme, InstallShield applies the theme to any dialogs that have a standard width and height: 374×266. If you switch from a wide theme to any other theme, InstallShield applies the theme to any dialogs that are 584×274.

Background Information on Which Dialogs Use the Exterior Dialog Style

By default, InstallShield uses the same style of dialog for all exterior dialogs: this style has an image on the left side of the dialogs. The InstallWelcome and SetupCompleteSuccess dialogs are examples of exterior dialogs.

If you add a custom exterior dialog to your project after you have selected a theme, InstallShield uses the selected theme for the new custom exterior dialog. However, if you later change the theme for your project, the theme may not be displayed properly in your custom exterior dialog. To resolve this type of issue, see Applying a Theme to a Custom Exterior Dialog.

Applying a Theme to a Custom Exterior Dialog



Project • *Dialog themes are available in Basic MSI projects.*

If you want to change the theme for your project and you also want to add a custom exterior dialog, it is easier to change the theme first, and then add the custom exterior dialog. Otherwise, the dialog may not appear properly; parts of newly selected theme's interior dialog images are added to your custom exterior dialog.

If you want to change the theme after you have added a custom exterior dialog to your project, you must add the name of your custom exterior dialog to the appropriate .theme file that is installed in the InstallShield Program Files folder. Then you can apply the theme to your project.



Task ***To change the theme after you have added a custom exterior dialog to your project:***

1. Close InstallShield.
2. In the InstallShield Program Files folder, find the .theme file for the theme that you want your dialogs to use. The default location is:

   C:\Program Files\InstallShield\2016\Support\Themes
3. Open the .theme file in a text editor such as Notepad. For example, if you want to use the Global theme, open the **Global.theme** file.
4. In the **<Include>** and **<Exclude>** sections of the file, add the following line:

   <Name>*NameOfDialog***</Name>**

   where *NameOfDialog* represents the name of your dialog.
5. Open your project in InstallShield.
6. In the View List under **User Interface**, click **Dialogs**.
7. In the **Dialogs** explorer, expand the **Themes** folder.
8. Right-click the theme that you want to use, and then click **Select**.

Adding a Logo or Other Image to the Exterior Dialogs



Project • *Dialog themes are available in Basic MSI projects.*

Exterior dialogs are dialogs that are displayed first and last by an installation—typically, the Welcome and Completion dialogs. The exterior dialogs for some themes have places for you to put the logo of your company or product. For example, the exterior dialogs of the Monitor theme and the Circles theme have a blank area on the left side of the dialog; you can add a bitmap control to these dialogs to customize your end-user interface.



Task ***To add an image to an exterior dialog:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **All Dialogs** folder, and then expand the dialog item for a dialog that should have the image.
3. Click a language under the dialog item that you expanded. The Dialog Editor in the center pane shows the dialog in the selected language.
4. In the **Controls** toolbar, click the **Bitmap** button.
5. In the Dialog Editor, click where you want the bitmap control to start, and holding the left mouse button down, drag the mouse to the place where you want it to end. Then release the left mouse button.
6. Configure the bitmap control's properties in the right pane.

   Click the **File Name** property, and then click the ellipsis button (**...**) to browse to the bitmap file that you want to use.

You may want to repeat the procedure for each language and each exterior dialog in your project. The exterior dialogs that are available by default in all new Basic MSI projects are:

- AdminWelcome
- InstallWelcome
- MaintenanceWelcome
- PatchWelcome
- SetupCompleteError
- SetupCompleteSuccess
- SetupInitialization
- SetupInterrupted
- SetupResume
- SplashBitmap

Available Themes and Corresponding Dialog Sizes



Project • *Some of the themes are available in both the Premier and Professional editions of InstallShield, and some are available in only the Premier edition.*

Some of the themes set the size of the dialogs to a standard width and height: 374×266. The wide themes set the dialog size to 584×274. The following table shows the size of each dialog, plus which editions of InstallShield contain the theme.

Table 4-6 • Dialog Themes in InstallShield

Name	InstallShield Edition	Dialog Width	Dialog Height
Circles Theme (Wide)	Premier and Professional	584	274
Classic Theme	Premier and Professional	374	266
Cooperation Theme (Wide)	Premier	584	274
Filmstrip Theme (Wide)	Premier	584	274
Global Theme	Premier and Professional	374	266
InstallShield Blue Theme	Premier and Professional	374	266
InstallShield Blue Theme (Wide)	Premier and Professional	584	274
InstallShield Silver Theme	Premier	374	266
Monitor Theme	Premier	374	266
Pastel Wheat Theme	Premier	374	266
Theater Theme (Wide)	Premier	584	274

Circles Theme (Wide)

Following are sample exterior and interior dialogs with the Circle Theme (Wide).

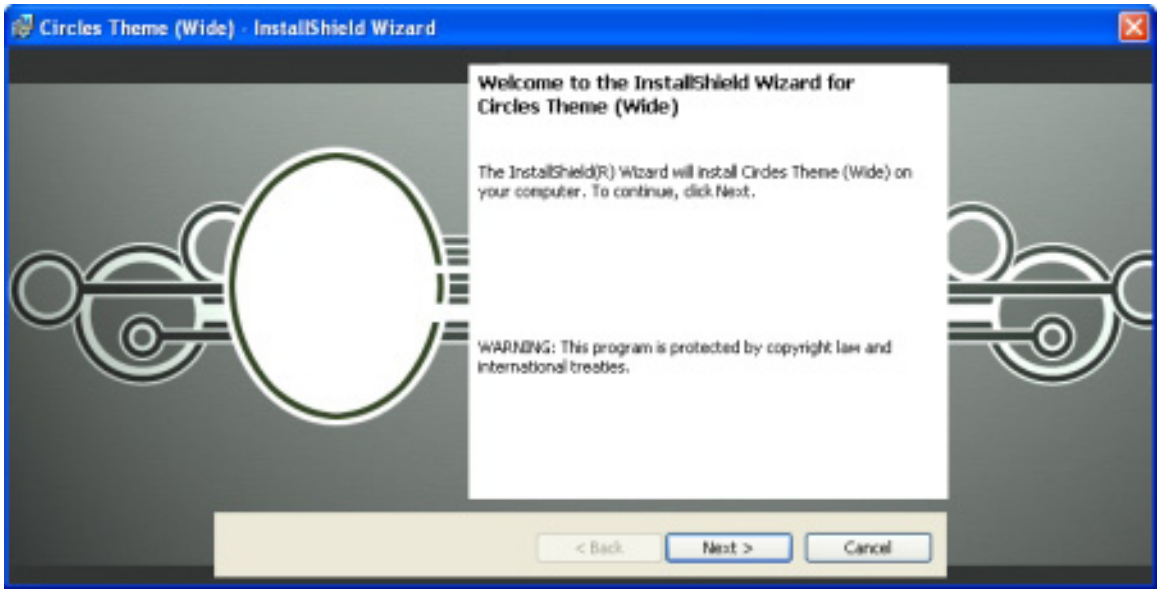


Figure 4-4: Sample Exterior Dialog with the Circle Theme (Wide)



Figure 4-5: Sample Interior Dialog with the Circle Theme (Wide)

To learn how to add your company or product logo to the exterior dialogs, see Adding a Logo or Other Image to the Exterior Dialogs.

Classic Theme

Following are sample exterior and interior dialogs with the Classic Theme.

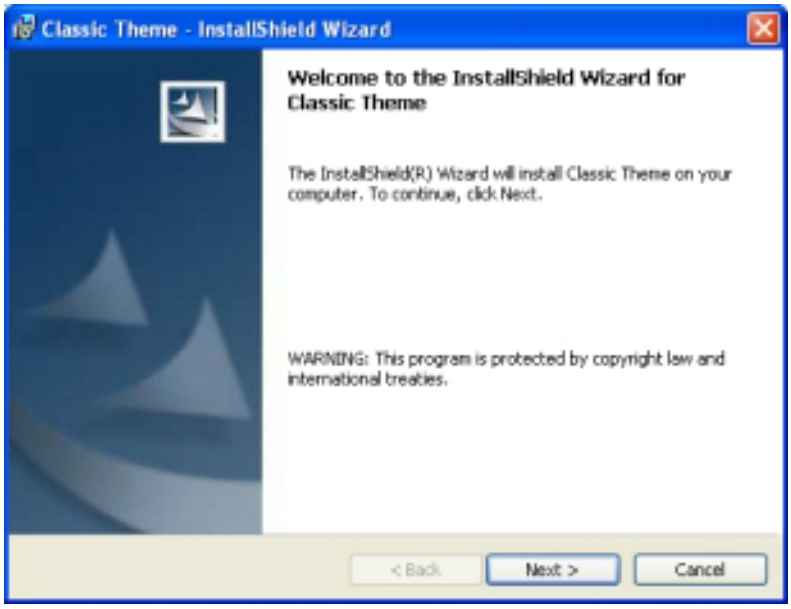


Figure 4-6: Sample Exterior Dialog with the Classic Theme

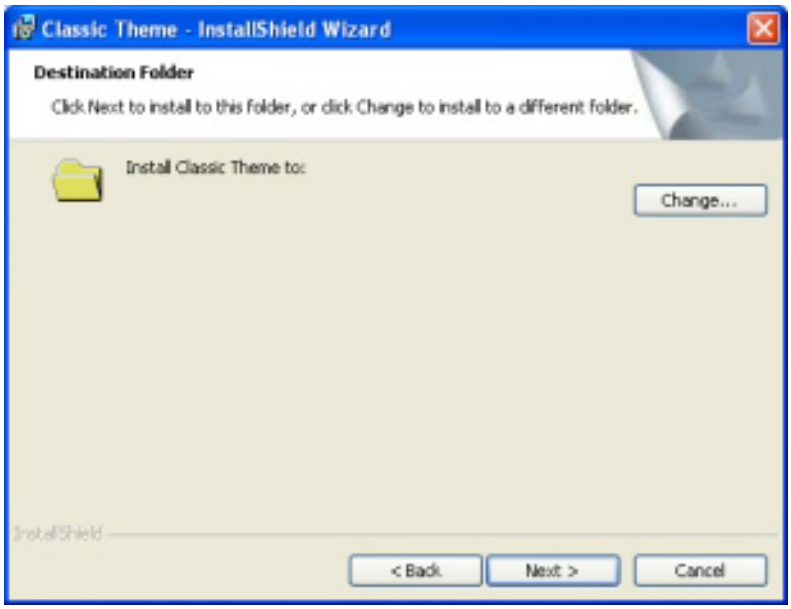


Figure 4-7: Sample Interior Dialog with the Classic Theme

Cooperation Theme (Wide)

Edition • *This theme is available in the Premier edition of InstallShield.*

Following are sample exterior and interior dialogs with the Cooperation Theme (Wide).



Figure 4-8: Sample Exterior Dialog with the Cooperation Theme (Wide)

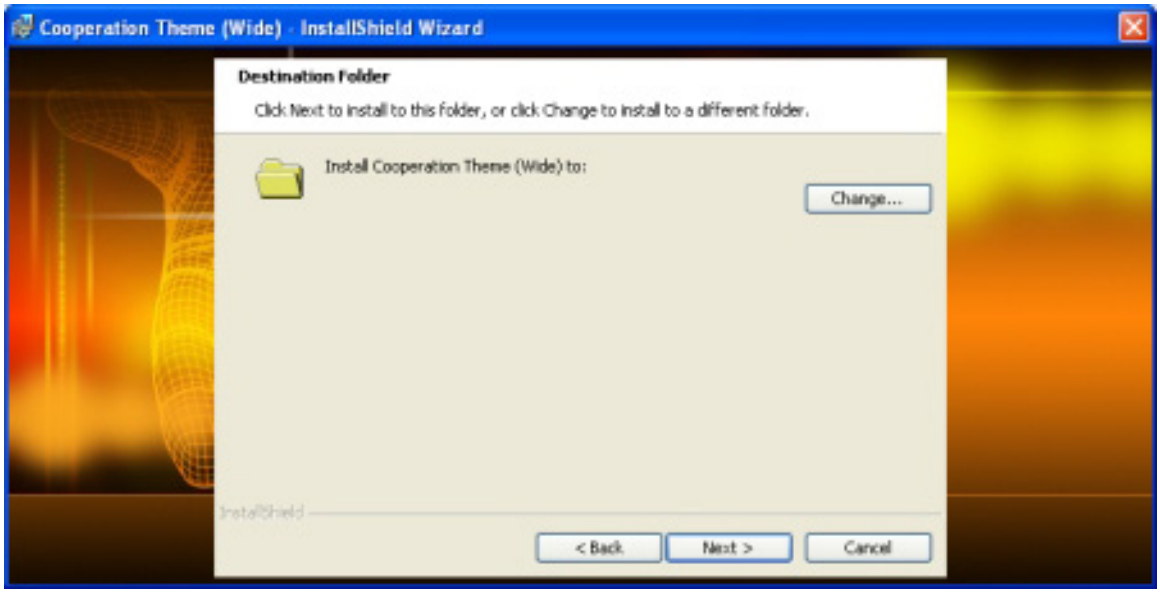


Figure 4-9: Sample Interior Dialog with the Cooperation Theme (Wide)

Filmstrip Theme (Wide)

Edition • *This theme is available in the Premier edition of InstallShield.*

Following are sample exterior and interior dialogs with the Filmstrip Theme (Wide).

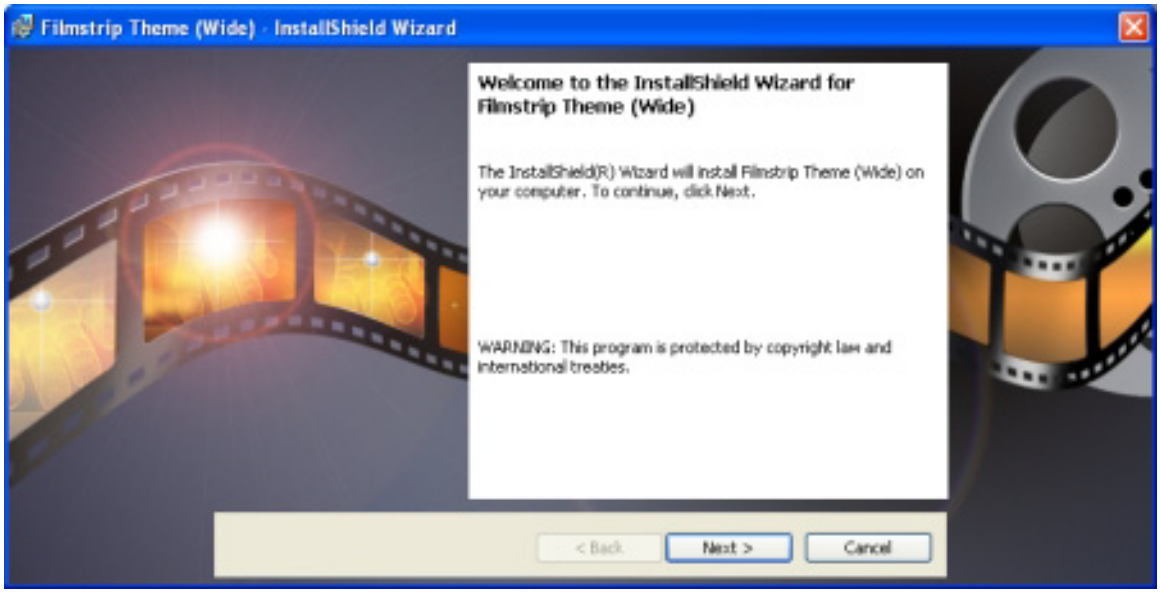


Figure 4-10: Sample Exterior Dialog with the Filmstrip Theme (Wide)

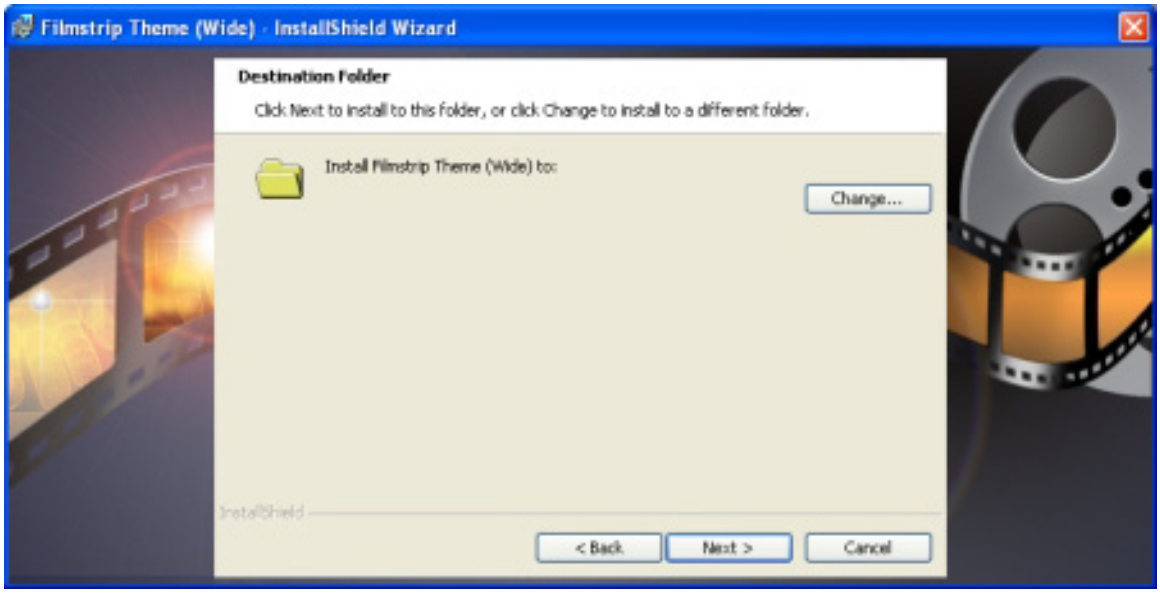


Figure 4-11: Sample Interior Dialog with the Filmstrip Theme (Wide)

Global Theme

Following are sample exterior and interior dialogs with the Global Theme.

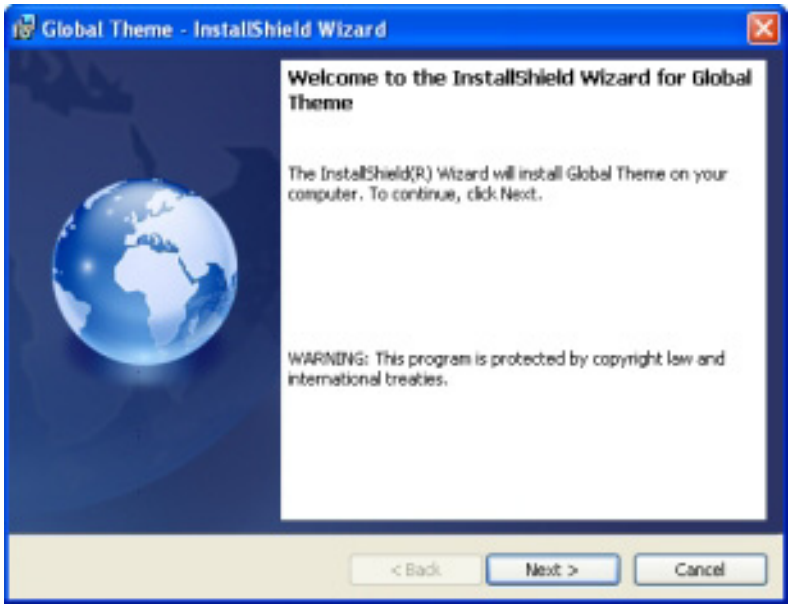


Figure 4-12: Sample Exterior Dialog with the Global Theme

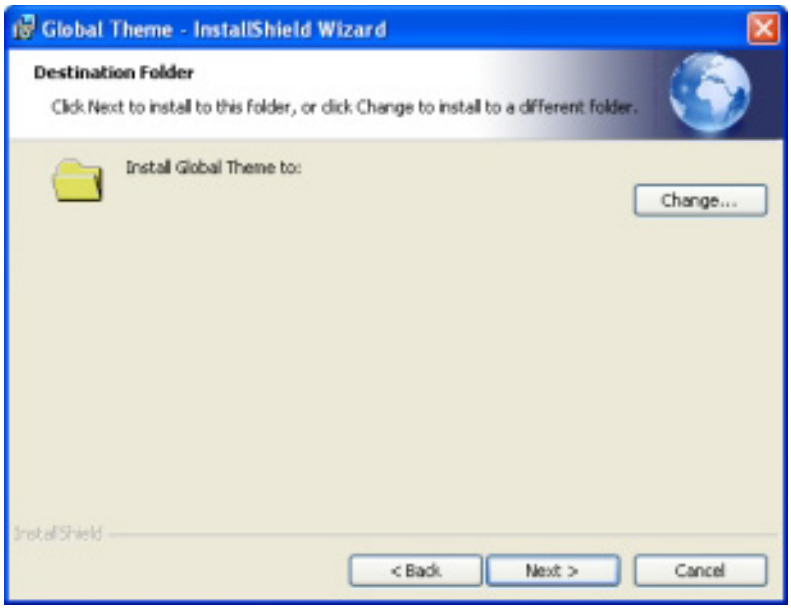


Figure 4-13: Sample Interior Dialog with the Global Theme

InstallShield Blue Theme

Following are sample exterior and interior dialogs with the InstallShield Blue Theme.

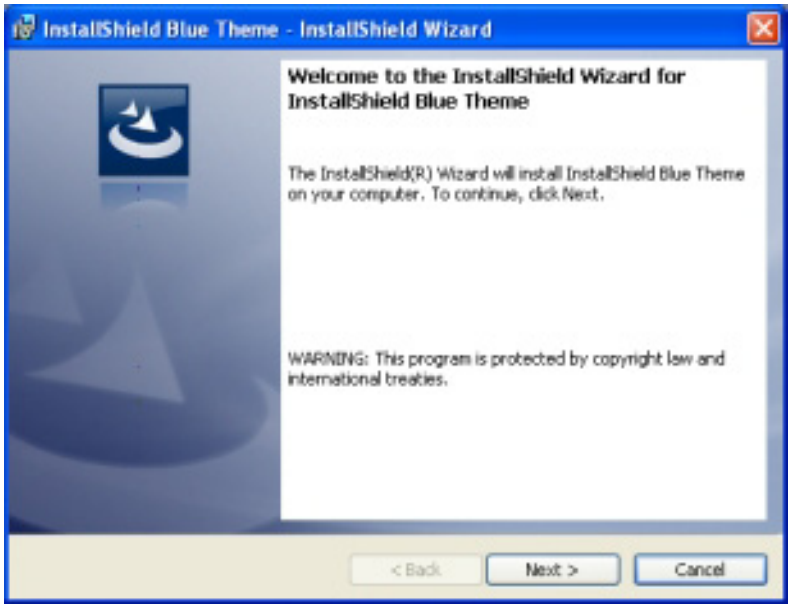


Figure 4-14: Sample Exterior Dialog with the InstallShield Blue Theme

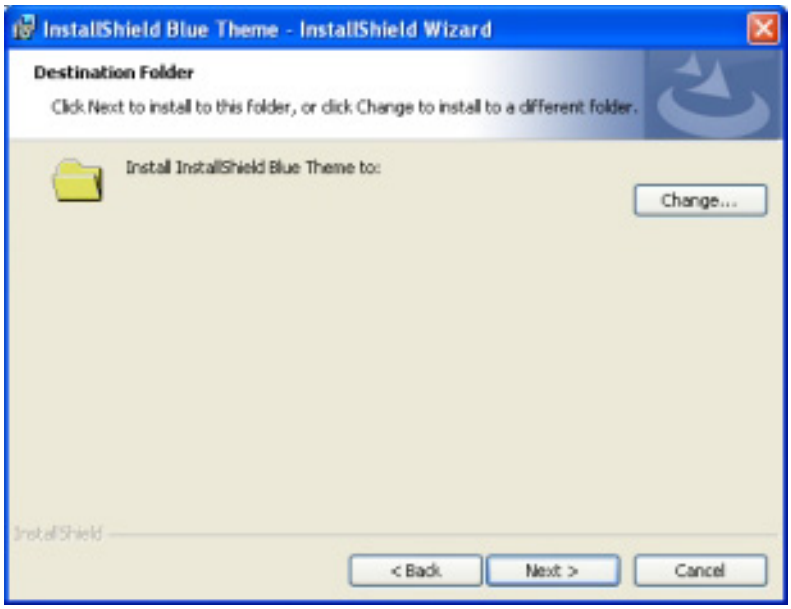


Figure 4-15: Sample Interior Dialog with the InstallShield Blue Theme

InstallShield Blue Theme (Wide)

Following are sample exterior and interior dialogs with the InstallShield Blue Theme (Wide).

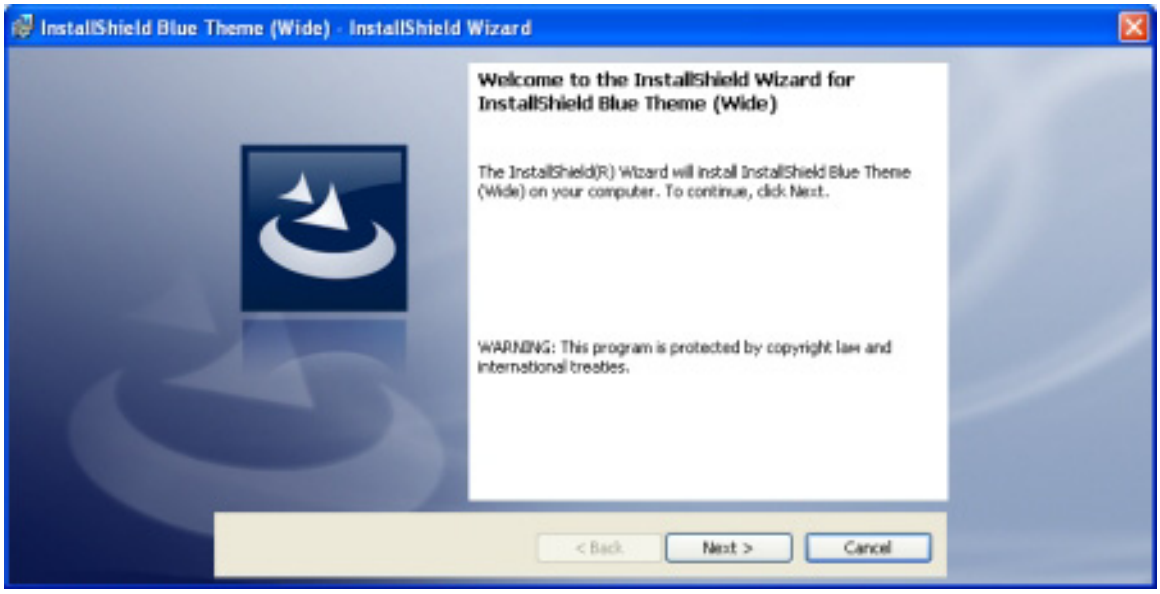


Figure 4-16: Sample Exterior Dialog with the InstallShield Blue Theme (Wide)



Figure 4-17: Sample Interior Dialog with the InstallShield Blue Theme (Wide)

InstallShield Silver Theme

Edition • *This theme is available in the Premier edition of InstallShield.*

Following are sample exterior and interior dialogs with the InstallShield Silver Theme.



Figure 4-18: Sample Exterior Dialog with the InstallShield Silver Theme



Figure 4-19: Sample Interior Dialog with the InstallShield Silver Theme

Monitor Theme

Edition • *This theme is available in the Premier edition of InstallShield.*

Following are sample exterior and interior dialogs with the Monitor Theme.

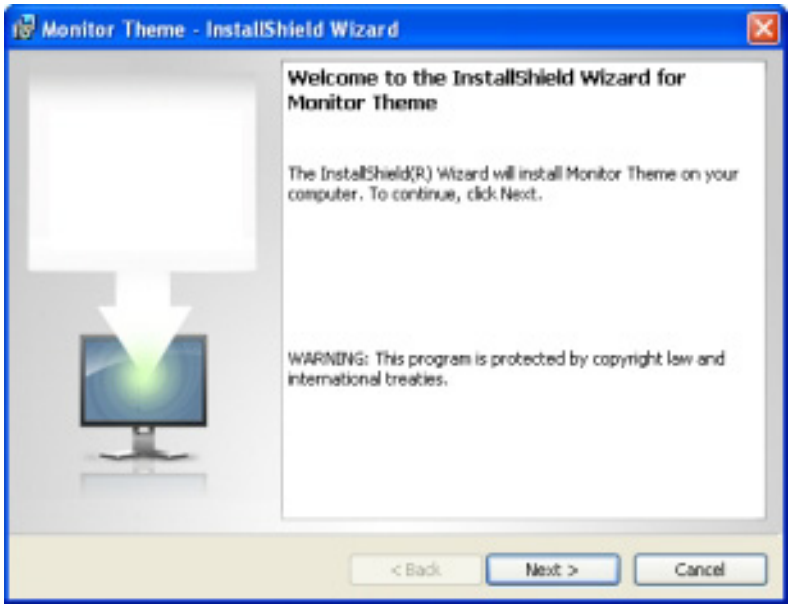


Figure 4-20: Sample Exterior Dialog with the Monitor Theme



Figure 4-21: Sample Interior Dialog with the Monitor Theme

To learn how to add your company or product logo to the exterior dialogs, see Adding a Logo or Other Image to the Exterior Dialogs.

Pastel Wheat Theme

Edition • *This theme is available in the Premier edition of InstallShield.*

Following are sample exterior and interior dialogs with the Pastel Wheat Theme.

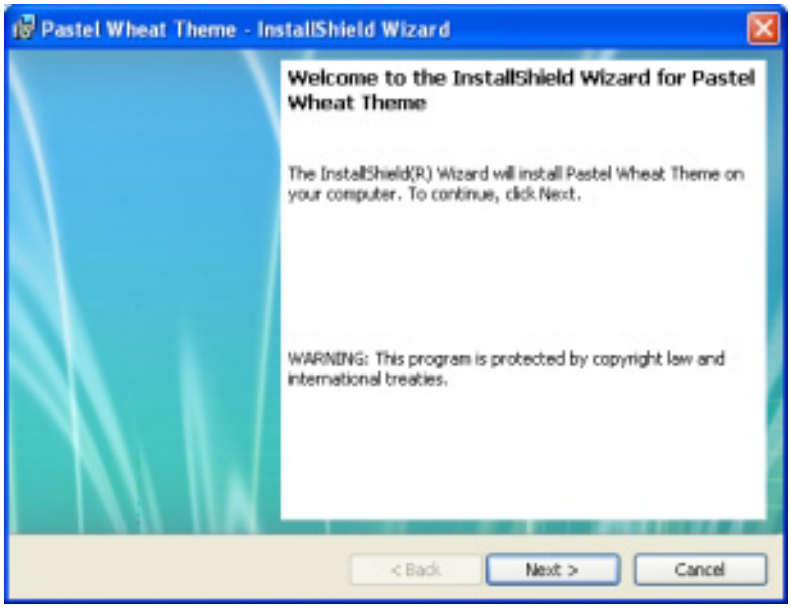


Figure 4-22: Sample Exterior Dialog with the Pastel Wheat Theme



Figure 4-23: Sample Interior Dialog with the Pastel Wheat Theme

Theater Theme (Wide)

Edition • *This theme is available in the Premier edition of InstallShield.*

Following are sample exterior and interior dialogs with the Theater Theme (Wide).

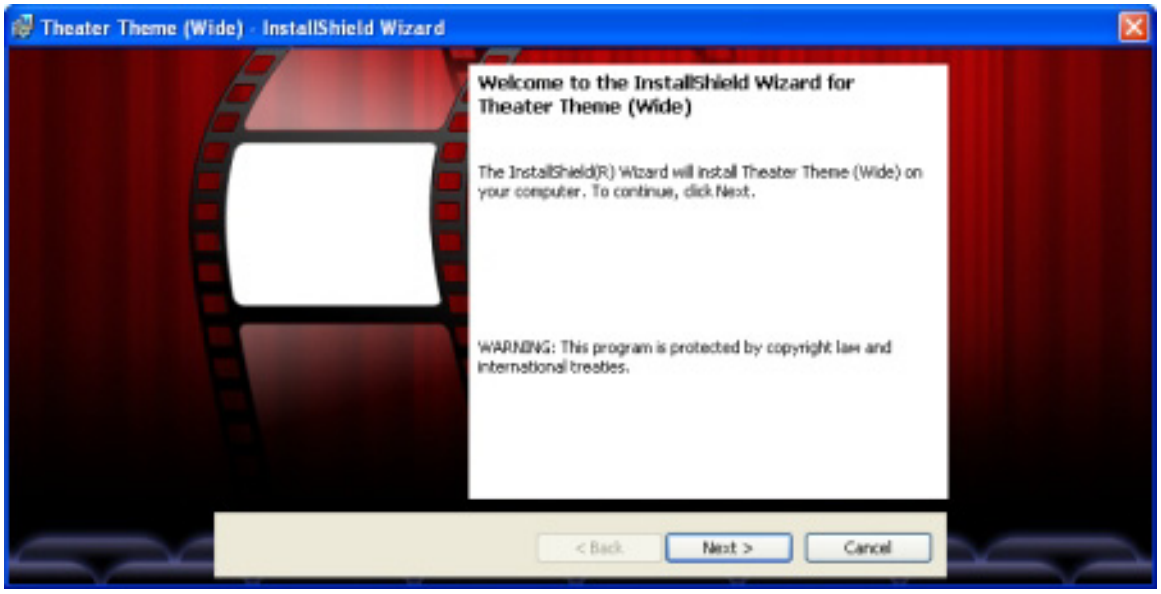


Figure 4-24: Sample Exterior Dialog with the Theater Theme (Wide)

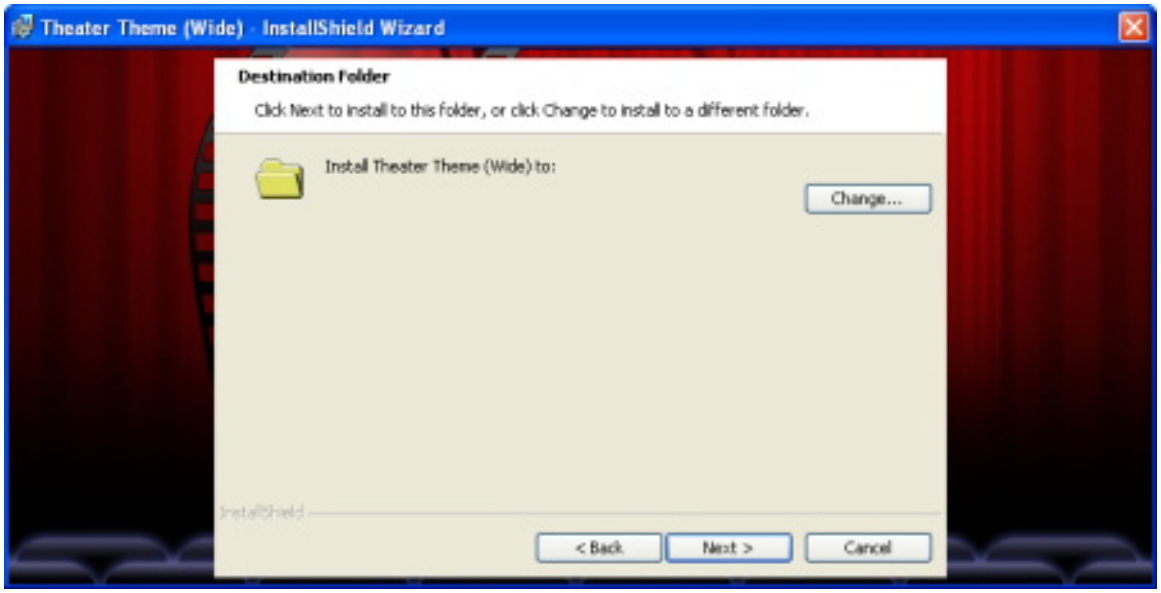


Figure 4-25: Sample Interior Dialog with the Theater Theme (Wide)

To learn how to add your company or product logo to the exterior dialogs, see Adding a Logo or Other Image to the Exterior Dialogs.

Dialog Support for Right-to-Left Languages

Edition • *The Premier edition of InstallShield includes dialog support for languages such as Arabic and Hebrew, which are read from right to left.*



***Project** • The following project types include dialog support for right-to-left languages:*

- *Basic MSI*
- *Merge Module*

InstallShield includes support for Arabic (Saudi Arabia) and Hebrew languages, which are written and read from right to left. All of the default end-user dialog strings are available in these languages.

Since these languages are read from right to left, InstallShield also includes support for mirroring Arabic and Hebrew dialogs; that is, InstallShield uses a right-to-left layout for Arabic and Hebrew dialogs. Thus, for example, buttons that are on the right side of dialogs in English and other left-to-right languages are moved to the left side of right-to-left-language dialogs. This occurs in the Dialog Editor pane in the Dialogs view of InstallShield; it also occurs at run time.

Reversed versions of the dialog images are displayed for the built-in dialog themes if appropriate. The reversed versions have *_mirror* in the image file name immediately before the .bmp or .jpg portion of the file name. For example, the name of an image for left-to-right languages might be **banner.jpg**; the name of the right-to-left equivalent for that image would be **banner_mirror.jpg**. These two files would be located in the same folder, and InstallShield would automatically use the **banner_mirror.jpg** file for the right-to-left-language versions of the dialogs. If an image should not be reversed, a *_mirror.jpg or *_mirror.bmp version of the image is not included, and the right-to-left versions of the dialogs do not show mirrored versions of the images.



***Tip** • If you use custom images for your run-time dialogs and your project includes support for right-to-left languages, you may need to create mirror-image versions for those right-to-left languages. You can preview the right-to-left layout of the dialogs in the Dialogs view to see how your custom images look. Add *_mirror.jpg or *_mirror.bmp versions of your images to the folder that contains the left-to-right versions of your custom images if appropriate.*

Launching a File Open Dialog



***Project** • This information applies to Basic MSI projects.*

InstallShield includes support for launching the Open dialog from one of the dialogs in your Basic MSI installation. End users click a browse button in one of your dialogs, and this launches the Open dialog. The Open dialog lets the end user browse for a file. When the end user selects the file and clicks the Open button, the Open dialog closes, and the installation writes the full path and file name in an edit field on the dialog. The installation also sets the value of the **IS_BROWSE_FILEBROWSED** property to the path and file name of the file that the end user selected.

When you incorporate support for the Open dialog in your installation, you can define several properties to specify the following functionality:

- You can specify the default path that is displayed in this dialog.
- You can specify the string that should be displayed in the **Files of type** drop-down list on the Open dialog.
- You can specify the file extensions of the files that should be displayed when the end user is browsing through folders to select a file. All files that have other file extensions are hidden and cannot be selected to open.
- You can specify the default file extension that the Open dialog should use. If the end user does not type an extension, the Open dialog appends this default extension to the file name.

Note the following requirements for this dialog:

- The Open dialog does not allow end users to create a new file. That is, if a file does not exist, an end user cannot manually type a new file name in the Open dialog.
- If the dialog that launches the Open dialog has more than one edit field control, the edit field control that will contain the full path to the file must have the lowest value for the Tab Stop property.



Task ***To add the Open dialog functionality to an end-user dialog:***

1. Add to your project a new MSI DLL custom action called **FileBrowse**:
   a. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
   b. Right-click the **Custom Actions** explorer, point to **New MSI DLL**, and then click **Stored in Binary table**. InstallShield adds a new custom action called NewCustomAction*N*, where *N* is a successive number.
   c. Change the name of the custom action to **FileBrowse**.
   d. In the pane on the right, configure the following settings for this custom action:
      - DLL Filename: **<ISRedistPlatformDependentFolder>\FileBrowse.dll**
      - Function Name: FileBrowse
      - Return Processing: Synchronous (Ignores exit code)
      - In-Script Execution: Immediate Execution
      - Execution Scheduling: Always execute

      For all other settings, leave the default values. The value of the MSI Type Number setting should be **65**.
2. Create or edit the dialog that should launch the Open dialog, and configure its behavior and layout:
   a. In the View List under **User Interface**, click **Dialogs**.
   b. In the **Dialogs** explorer, expand the **All Dialogs** folder.
   c. Either select an existing dialog or create a new dialog.

      Note that if you select an existing dialog and it has more than one edit field control, the edit field control that will contain the full path to the file must have the lowest value for the **Tab Stop** property.
   d. Under this dialog, click the language whose layout you want to configure.
   e. Add the edit field control that will contain the full path and file name that the end user selects at run time. When InstallShield prompts you for the property name associated with the control, enter **IS_BROWSE_FILEBROWSED**.
   f. Next to the edit field control, add a push button control. This is the button that will launch the Open dialog.
   g. Select the push button control and then edit its properties in the grid on the right as needed. For example, to specify the text that you want to be displayed on the button, add a value for the **Text** property.
   h. In the **Dialogs** explorer, click the **Behavior** item under the dialog that you are configuring.
   i. In the center pane that lists the dialog controls, click the push button control that you just created. Its settings are displayed in the right pane.

j. In the **Events** setting, click the **New Event** button, and then click **DoAction**. InstallShield adds a new set of rows under the **Events** setting.

k. In the **DoAction** setting, enter the following:

1

l. In the **Action** subsetting, specify the following action name:

FileBrowse

m. In the **Events** setting, click the **New Event** button, and then click **SetProperty**. InstallShield adds a new set of rows under the **Events** setting.

n. In the **SetProperty** setting, enter the following:

1

o. In the **Property** subsetting, specify the following action name:

IS_BROWSE_FILEBROWSED

p. In the **Value** subsetting, specify the following action name:

[IS_BROWSE_FILEBROWSED]



***Note** • The SetProperty Event refreshes the MSI property value displayed in the Edit field control on the dialog, so that the displayed MSI property value is updated after selecting a file in the file browse window.*

3. Configure several properties to specify behavior of the Open dialog and the dialog that launches the Open dialog:

a. In the View List under **Behavior and Logic**, click **Property Manager**.

b. Find the **IS_BROWSE_FILEBROWSED** property. Its default value is 0. Do one of the following:

- To leave the edit field control blank in your dialog when the end user first displays it, right-click the row that has the **IS_BROWSE_FILEBROWSED** property and then click **Delete Property**.
- To display a default path and file name in the edit field control, change the value of the **IS_BROWSE_FILEBROWSED** property to the path and file name.



***Note** • If you do not manually change the value of this property or delete this property, the default value for the edit field control on the dialog that launches the Open dialog is set to 0.*

c. Optionally, add a property called **IS_BROWSE_FILEEXT**, and set its value to a filter string that identifies the file extensions that should be displayed when the end user is browsing through folders to select a file. All files that have other file extensions are hidden and cannot be selected to open.

A filter string can be a combination of valid file name characters and the asterisk (*) as a wild-card character.

To specify multiple file extensions, separate each with a semicolon. Do not include spaces. For example, to let end users select .exe and .dll files, enter the following string as the value of the **IS_BROWSE_FILEEXT** property:

***.exe;*.dll**

In this example, the Open dialog lets end users select .exe and .dll files. It hides all other file types.

If you do not set this property, the Open dialog lets end users select any file type.

d. Optionally, add a property called **IS_BROWSE_FILETYPE**, and set its value to the string that you want to be displayed in the **Files of type** drop-down list on the Open dialog. Note that only one option can be displayed in this drop-down list.

For example, if you want end users to be able to select .txt or .doc files, enter the following string as the value of the **IS_BROWSE_FILETYPE** property:

Text Files (*.txt); Word documents (*.doc)

If you do not set this property, the **Files of type** drop-down list in the Open dialog is blank.

e. Optionally, add a property called **IS_BROWSE_DEFAULTEXTENSION**, and set its value to the default file extension that the Open dialog should use. If the end user does not type an extension, the Open dialog appends this default extension to the file name. For example, to use .exe as the default file extension, enter the following string as the value of the **IS_BROWSE_DEFAULTEXTENSION** property:

exe

f. Optionally, add a property called **IS_BROWSE_INITIALDIR**, and set its value to the default path that the Open dialog should use. For example, to use **C:\Program Files\My Product**, enter the following string as the value of the **IS_BROWSE_INITIALDIR** property:

C:\Program Files\My Product

At run time, when the end user clicks the new push button control, the Open dialog opens. The end user can browse to and select a file. The installation sets the value of the **IS_BROWSE_FILEBROWSED** property to the path and file name of the file that the end user selected, and then it displays that path and file name in the edit field control on the dialog that launched the Open dialog.

Requiring End Users to Scroll Through the EULA in the LicenseAgreement Dialog



Project • *This information applies to Basic MSI projects.*

InstallShield includes support for disabling the Next button on the LicenseAgreement dialog until the end user reaches the end of the End-User License Agreement (EULA) text in the scrollable EULA control through one of the following methods:

- Using the scroll bar.
- Pressing PAGE DOWN when the scrollable EULA control has focus.
- Pressing CTRL+PAGE DOWN when the scrollable EULA control has focus.
- Pressing the DOWN ARROW key when the scrollable EULA control has focus.
- Right-clicking the scroll bar and then clicking Bottom.

The end user must also select the **I accept the terms in the license agreement** option before the Next button is enabled.

The LicenseAgreement dialog requires end users to select the **I accept** option by default. If you also want to require end users to reach the end of the EULA text, perform the following task.

Task ***To require end users to reach the end of the EULA text in the scrollable EULA control:***

1. Add to your project a new MSI DLL custom action called **WatchScroll**:
   a. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
   b. Right-click the **Custom Actions** explorer, point to **New MSI DLL**, and then click **Stored in Binary table**. InstallShield adds a new custom action called NewCustomAction*N*, where *N* is a successive number.
   c. Change the name of the custom action to **WatchScroll**.
   d. In the pane on the right, configure the following settings for this custom action:
      - DLL Filename: **<ISRedistPlatformDependentFolder>\EulaScrollWatcher.dll**
      - Function Name: WatchScroll
      - Return Processing: Asynchronous (Waits for exit code)
      - In-Script Execution: Immediate Execution
      - Execution Scheduling: Always execute

      For all other settings, leave the default values. The value of the MSI Type Number setting should be **129**.
2. Edit the LicenseAgreement dialog so that it launches the appropriate custom action:
   a. In the View List under **User Interface**, click **Dialogs**.
   b. In the **Dialogs** explorer, expand the **All Dialogs** folder, and expand the **LicenseAgreement** item, and then click **Behavior**.
   c. In the center pane that lists the LicenseAgreement controls, click the ScrollableText control named **Memo**. This is the control that contains the text of the EULA. Its settings are displayed in the right pane.
   d. In the **Events** setting, click the **New Event** button, and then click **DoAction**. InstallShield adds a new set of rows under the **Events** setting.
   e. In the **DoAction** setting, enter the following:

      Not LicenseViewed AND Not ISLicenseWatching
   f. In the **Action** subsetting, specify the following action name:

      WatchScroll
   g. In the center pane that lists the LicenseAgreement controls, click the push button control named **Next**.
   h. In the **Conditions** setting, click the **New Condition** button, and then click **Disable**. InstallShield adds a **Disable** setting.
   i. In the **Disable** setting, enter the following:

      AgreeToLicense <> "Yes" OR Not LicenseViewed
   j. In the **Conditions** setting, click the **New Condition** button, and then click **Enable**. InstallShield adds an **Enable** setting.

k. In the **Enable** setting, enter the following:

AgreeToLicense = "Yes" AND LicenseViewed

At run time, the installation monitors the EULA control in an asynchronous custom action. While the custom action is running, the **ISLicenseWatching** property is set. Once the custom action is finished, it removes the **ISLicenseWatching** property. This avoids the hourglass flicker otherwise seen when end users scroll through the EULA text. Once the end user reaches the bottom of the EULA text, the installation sets the **LicenseViewed** property and manually enables the Next button if necessary according to the Next button's conditions. The Next button is found via its text as stored in the **Control** table; therefore, you can use the aforementioned procedure with any language transform as long as the control is named **Next**.

Adding a Print Button to a Dialog

In Basic MSI projects that are created in InstallShield X or later, the LicenseAgreement dialog includes a Print button. This button enables the end user to print the content of the dialog's ScrollableText control. This button's event executes the custom action ISPrint, which is included in a new Basic MSI project. Following are directions for adding a Print button to another dialog, and to an existing project that was created with InstallShield DevStudio 9.0 or earlier.

Adding a Print Button to Another Dialog

For the custom action ISPrint to work correctly with a dialog other than LicenseAgreement, you must set the value of the user-defined Windows Installer property **IS_PRINT_DIALOG** to the name of the dialog. (If **IS_PRINT_DIALOG** is not an existing property, ISPrint prints the content of the LicenseAgreement dialog's ScrollableText control.)



Task ***To add a Print button to another dialog:***

1. Create a button control in the dialog and optionally set its **Text** property to **&Print**. For details, see Editing Dialog Layout in Basic MSI Projects.
2. Add a DoAction event to the Print button, and in the event's Action subsetting, select **ISPrint**. For details, see Triggering Control Events in Basic MSI Dialogs.
3. Modify the value of **IS_PRINT_DIALOG** from the events of the Back and Next buttons of the dialog and its next and previous dialogs:
   a. Determine which dialog is displayed before the dialog to which you are adding a Print button. You can do this by either checking the argument of the NewDialog event for the dialog's Back button, or viewing next dialog order in the expanded Custom Actions and Sequences view.
   b. Add a SetProperty event. In the **SetProperty** setting, specify the following as the event condition:

      1

   c. In the **Property Name** setting, specify the following:

      IS_PRINT_DIALOG

   d. In the **Value** setting, specify the name of the dialog to which you are adding a Print button
   e. If a Print button is included on any dialog that is displayed after the dialog to which you are adding a Print button, do the following:

i. Determine which dialog is displayed after the dialog to which you are adding a Print button. You can do this by either checking the argument of the NewDialog event for the dialog's Next button, or viewing the next dialog order in the expanded Custom Actions and Sequences view.

ii. Add a SetProperty event to that next dialog's Back button, and set its **Property Name** setting to **IS_PRINT_DIALOG**, and its **Value** setting to the name of the dialog to which you are adding a Print button.

If the next dialog does not have a Print button, or if it is the LicenseAgreement dialog, add a SetProperty event to the Next button of the dialog to which you are adding a Print button, and set its **Property Name** setting to **IS_PRINT_DIALOG**, and its **Value** setting to **LicenseAgreement**.

f. If a Print button is included on any dialog that is displayed before the dialog to which you are adding a Print button, and the previous dialog does not have a Print button or it is the LicenseAgreement dialog, add a SetProperty event to the Back button of the dialog to which you are adding a Print button, and set its **Property Name** setting to **IS_PRINT_DIALOG**, and its **Value** setting to **LicenseAgreement**.

Adding a Print Button to an Existing Project



Task ***To add a Print button to an existing project that was created with InstallShield DevStudio 9.0 or earlier:***

1. Create the ISPrint custom action:
   - **a.** Launch the Custom Action Wizard.
   - **b.** In the **Basic Information** panel's **Name** box, enter **ISPrint**.
   - **c.** In the **Action Type** panel's **Type** box, select **Call a function in a Windows Installer dynamic-link library**.
   - **d.** In the **Action Parameters** panel, click the **Browse** button and browse to the file **SetAllUsers.dll** in the InstallShield folder's **Redist\Language Independent\i386** subfolder. Click **Open**.

     If your project is configured to use path variables, InstallShield uses the predefined path variable **<ISRedistPlatformDependentFolder>** for part of the path.
   - **e.** In the **Action Parameters** panel's **Target** box, enter **PrintScrollableText**.
   - **f.** Complete the wizard, accepting all remaining default settings.
2. Create a button control in the dialog and optionally set its **Text** property to **&Print**. For details, see Editing Dialog Layout in Basic MSI Projects.
3. Add a DoAction event to the Print button, and in the event's **Action** subsetting, select **ISPrint**. For details, see Triggering Control Events in Basic MSI Dialogs.
4. Create the Windows Installer property **IS_PRINT_DIALOG** and set its value to the name of the dialog. For details, see Creating Properties in Windows Installer–Based Projects.



Note *• If the ISPrint custom action is executed by a control event, the custom action's logging information cannot be recorded to the installer log in the usual manner (because of a Windows Installer limitation); the information is logged to the values of properties that have the form ISPrintLogmNoten.*

Minimizing Reboots on Windows Vista and Later Systems

Windows Logo • *Restarting the system after an installation is inconvenient for end users. One of the Windows logo program requirements is that all installations must contain an option that enables end users to automatically close applications and attempt to restart them after the installation is complete.*

To support this requirement, all Basic MSI projects include the MsiRMFilesInUse dialog by default. An installation displays the MsiRMFilesInUse dialog on a Windows Vista or later system if one or more files that need to be updated are currently in use during the installation. The dialog contains two options to allow end users to specify how to proceed:

- End users can choose to have the installation close the applications that are using those files and then attempt to restart the applications after the installation is complete.
- End users can avoid closing the applications. A reboot will be required at the end of the installation.

For the best end-user experience, your application should be instrumented to use the Restart Manager API; doing so allows the Restart Manager to effectively pause and resume your application exactly where the end user left it. For detailed information, see About Restart Manager and the other Restart Manager documentation on the MSDN Web site.

Dialog Controls

Whether your dialog is part of a Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, or Merge Module project, you can use many of the same controls to modify the layout and behavior of your predefined and custom dialogs.

Table 4-7 • Dialog Controls

Control Type	Project Type	Description
Billboard	Basic MSI, Merge Module	A billboard control is used to display data that can be updated in response to control events. You can use a billboard, for example, to display the progress of a protracted custom action. ***Project*** • *If your installation includes a* **Setup.exe** *launcher, you can have the launcher display billboards during the file transfer process; this is an alternative to the billboard control, which is a Windows Installer control. This* **Setup.exe** *billboard support is available in Basic MSI, InstallScript, and InstallScript MSI installations. To learn more about this type of billboard, see Displaying Billboards.*
Bitmap	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A bitmap control displays an image.

Table 4-7 • Dialog Controls (cont.)

Control Type	Project Type	Description
Check box	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A check box control displays a check box that end user can select or clear.
Combo box	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A combo box control is a box that contains a drop-down list of predefined values; the box is also an edit field in which an end user can enter a value.
Dialog	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A dialog control is associated with a series of settings that you configure.
Directory combo	Basic MSI, Merge Module	A directory combo is used in conjunction with a directory list and a path edit control to create a browse dialog. The directory combo displays the list of drives mapped on the current system.
Directory list	Basic MSI, Merge Module	A directory list is used in conjunction with a directory combo and a path edit control to create a browse dialog. The directory list displays the folders below the drive that is selected in the directory combo control, and it populates the value of the path edit control.
Edit Field	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	An edit field control is a text box in which end users can enter a string or an integer.
Group box	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A group box can be used to enclose various controls in one area. A group box also provides a label that you can use to express the relationship between the controls that are contained within it.

Table 4-7 • Dialog Controls (cont.)

Control Type	Project Type	Description
Hyperlink	Basic MSI, Merge Module	The hyperlink control displays an HTML link; clicking the link at run time opens a page in the default browser on the target system. ***Important*** • *Windows Installer 5 and later include support for the hyperlink control.* ***Project*** • *To learn how to add an HTML control to a dialog in an InstallScript project or in an InstallScript MSI project, see Using an HTML Control on a Dialog.*
Icon	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The icon control displays a picture of an icon.
Line	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The line control creates lines of adjustable length and thickness. You can use lines on a dialog to separate areas of the dialog or add graphical touches.
List box	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The list box control is a standard list box that lets end users select a single option from a list of predetermined options.
List view	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The list view control displays a single column of options; an icon is displayed next to each option.
Masked edit	Basic MSI, Merge Module	The masked edit control is essentially an edit field control that lets end users enter information in a specified format.
Path edit	Basic MSI, Merge Module	A path edit control is used in conjunction with a directory combo control and a directory list control to create a browse dialog. The path edit displays the complete path that is assembled from selections that the end user makes in the directory combo and directory list, along with or instead of edits that the end user makes.

Table 4-7 • Dialog Controls (cont.)

Control Type	Project Type	Description
Progress bar	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A progress bar is a dynamic, graphical bar that fills up in response to control events.
Push button	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A push button control is a button that carries out a command when an end user clicks it.
Radio button	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A radio button control is one of two or more mutually exclusive options that end users can select. A radio button must be inserted into a radio button group; it functions as part of that control.
Radio button group	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A radio button group control is a container for radio button controls. A radio button group and its radio buttons behave as a single control.
Scrollable text	Basic MSI, Merge Module	A scrollable text control displays a long string of text that cannot fit on the dialog; the control includes scroll bars that enable end users to scroll through the text. The LicenseAgreement dialog is an example of a dialog that typically contains a scrollable text control.
Selection tree	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A selection tree is a special control that lets end users change the selection state of your features, like in the CustomSetup dialog.
Text area	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	A text area control displays text.
Volume cost list	Basic MSI, Merge Module	A volume cost list control displays the disk space requirements that are associated with each volume or drive.
Volume select combo	Basic MSI, Merge Module	A volume select combo control enables the end user to select from an alphabetical list of volumes, or drives.

Billboard Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

If your installation includes a **Setup.exe** *launcher, you can have the launcher display billboards during the file transfer process; this is an alternative to the billboard control, which is a Windows Installer control. This* **Setup.exe** *billboard support is available in Basic MSI, InstallScript, and InstallScript MSI installations. To learn more about this type of billboard, see Displaying Billboards.*

A billboard control is used to display data that can be updated in response to control events. Billboards can contain other controls for displaying this information, but they must be static controls—including text, bitmaps, and icons—that are not linked to a Windows Installer property. You can use a billboard, for example, to display the progress of a protracted custom action.

Billboards are not fully supported in the Dialog Editor. In order to have the billboard interact with Windows Installer actions and display other controls, you must make changes to the **Billboard** and **BBControl** tables in your project; you can use the Direct Editor view to modify these tables.

When you select a billboard control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-8 • Settings for a Billboard Control

Setting	Description
Name	Enter a name for this billboard. The name must be unique among all of the controls in your project.
Base Text Style	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).

Table 4-8 • Settings for a Billboard Control (cont.)

Setting	Description
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	This setting contains the text that is used for the initial value of the billboard's control (see the **BBControl** table). When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. If the billboard's control contains a bitmap or icon, this value must be a foreign key into the **Binary** table to the file that is initially displayed on the control.
Text Style	Select the font style, size, and color (if available) in which you want the billboard's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.

Table 4-8 • Settings for a Billboard Control (cont.)

Setting	Description
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Bitmap Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A bitmap control displays an image.

When you select a bitmap control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-9 • Settings for a Bitmap Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this bitmap. The name must be unique among all of the controls in your project.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.

Table 4-9 • Settings for a Bitmap Control (cont.)

Setting	Project Type	Description
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Select True if this is the only control on the dialog that you want to be the default control, which means that it will be activated when the end user presses the ENTER key. The Next or OK button is usually the default control.
File Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter the path to and the name of the image file that you want to use for this control, or click the ellipsis button (...) in this setting to browse to it. InstallShield adds the file to your release at build time. The file must be stored as a binary resource in the installation.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Stretch to Fit	Basic MSI, Merge Module	If you want the image to be resized to fill the area of the control, select True. To have the image centered if it is smaller than the control or to have the image cropped if it is bigger than the control, select False.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.

Table 4-9 • Settings for a Bitmap Control (cont.)

Setting	Project Type	Description
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Check Box Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A check box control displays a check box that end user can select or clear.

When you select a check box control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.



Project • *(For Basic MSI and Merge Module projects) When you first draw this type of control on a dialog, InstallShield prompts you for the name of a Windows Installer property. InstallShield uses the name that you enter as the value for this control's Property setting. At run time, the installation sets the value of this property based on the end user's selection. For more details, see Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties.*

Table 4-10 • Settings for a Check Box Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this check box. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting. This setting is enabled if Text is selected for the Control Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Control Style	Basic MSI, Merge Module	Specify whether this control is marked with a text label, an icon, or a bitmap.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.

Table 4-10 • Settings for a Check Box Control (cont.)

Setting	Project Type	Description
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
File Name	Basic MSI, Merge Module	This setting is enabled if you select Bitmap or Icon for the Control Style setting. Enter the path to and the name of the image file that you want to use for this control, or click the ellipsis button (...) in this setting to browse to it. InstallShield adds the file to your release at build time. The file must be stored as a binary resource in the installation. This setting is enabled if Bitmap or Icon are selected for the Control Style setting.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.).
Icon Size	Basic MSI, Merge Module	Assuming that your icon file has more than one resource, specify the size of the image that you want to use for the control. The **Use first image** option causes the first image in the file to be displayed and makes it stretch to fit the size of the control, regardless of what you specify for the Stretch to Fit setting. This setting is enabled if Bitmap or Icon are selected for the Control Style setting.
Indirect Property	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).

Table 4-10 • Settings for a Check Box Control (cont.)

Setting	Project Type	Description
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Property	Basic MSI, Merge Module	Enter the name of a property that is set when an end user selects this check box. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the Value setting.
Property Is Integer	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property for this control (specified in the Property setting) has an integer value, select True. If the property's value is a string, select False.
Push Button	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To turn this check box into a push button control, select True; otherwise, select False.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Stretch to Fit	Basic MSI, Merge Module	If you want the image to be resized to fill the area of the control, select True. To have the image centered if it is smaller than the control or to have the image cropped if it is bigger than the control, select False. This setting is enabled if Bitmap or Icon are selected for the Control Style setting.

Table 4-10 • Settings for a Check Box Control (cont.)

Setting	Project Type	Description
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This setting contains the text that is used for the control's label. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. This setting is enabled if Text is selected for the Control Style setting.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property. This setting is enabled if Text is selected for the Control Style setting.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).

Table 4-10 • Settings for a Check Box Control (cont.)

Setting	Project Type	Description
Value	Basic MSI, Merge Module	When the check box is selected, the property that is specified for the Property setting is set to this value.
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Combo Box Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A combo box control is a box that contains a drop-down list of predefined values; the box is also an edit field in which an end user can enter a value.

When you select a combo box control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.



Project • *(For Basic MSI and Merge Module projects) When you first draw this type of control on a dialog, InstallShield prompts you for the name of a Windows Installer property that identifies all of the items that belong to this combo box. InstallShield uses the name that you enter as the value for this control's Property setting. At run time, the installation sets the value of this property based on the end user's selection. For more details, see Working with Windows Installer*

and Advanced UI or Suite/Advanced UI Properties. (Note that Windows Installer combo boxes allow the end user to select only a single item.)

Table 4-11 • Settings for a Combo Box Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this combo box. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Code Page	Basic MSI, Merge Module	If you want the control to use fonts from the code page that is defined in the installation's package, select Database. If you want the control to use fonts from the target system's default code page, select User's System.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Drop-Down List	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want the control to be a drop-down list, select True. If you want the control to be an editable combo box, select False.

Table 4-11 • Settings for a Combo Box Control (cont.)

Setting	Project Type	Description
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.  ***Tip*** • *Be sure to set the Height setting to a large-enough number; this setting specifies the height of the drop-down list portion of the combo box control.*
Indirect Property	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Items	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To specify the options that you want to be listed in this control, click the ellipsis button (...) in this setting; this opens the List Items dialog box. To add a new item to the list, click the Add button on the List Items dialog box. For each item, you must enter the text that is displayed in the combo box, and a value, which is the value that is assigned to the property if this item is selected.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.

Table 4-11 • Settings for a Combo Box Control (cont.)

Setting	Project Type	Description
Max. Length	Basic MSI, Merge Module	Specify how many characters the end user can enter in the combo box.
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Property	Basic MSI, Merge Module	Enter the name of a property that is set when the end user enters a value into or selects one from this combo box. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the value of the default selection (specified in the Items setting).
Property Is Integer	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property for this control (specified in the Property setting) has an integer value, select True. If the property's value is a string, select False. Make sure that all of the values in the Item setting are either integers or strings, depending on the value that you select in the Property Is Integer setting.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sorted	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Selecting True causes the items in the combo box to be sorted alphabetically. A value of False makes them retain the order that you set for each item in the List Items dialog box.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.

Table 4-11 • Settings for a Combo Box Control (cont.)

Setting	Project Type	Description
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Dialog Control



Project • *This information applies to the following project types:*

- *Basic MSI*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

When you select a dialog in the Dialog Editor of the Dialogs view, InstallShield displays the following settings in the right pane.

Unlike its controls, you cannot change the name of a dialog in the Dialog Editor. To change the name, right-click the dialog in the Dialogs view and click Rename.

Table 4-12 • Settings for a Dialog Control

Setting	Project Type	Description
Caption	Basic MSI, Merge Module	Specify the name that you want to use as the title for the dialog. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Comment	Basic MSI, Merge Module	Enter comments for the dialog. Your comments are saved in the project file for your reference and are not used in the installation at any time.
Custom Palette	Basic MSI, Merge Module	A custom palette is necessary only if the dialog contains images that use a color palette that is different from the default one that is created by Windows Installer. If the dialog does not contain any images that use a different color palette, leave this value as False.
Error Dialog	Basic MSI, Merge Module	If this dialog serves as an error dialog, select True.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the height of the dialog in installer units (1/12 of the height of the system font).
Keep Modeless	Basic MSI, Merge Module	When this value is set to True and this dialog is spawned through a DoAction event, all other dialogs remain. If the Keep Modeless setting is False in this case, the other dialogs are not displayed.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the percentage from the left edge of the screen where you want the dialog to be placed. A value of 50 centers the dialog horizontally. This value is ignored if this dialog is part of an installation wizard, and the previous dialog was in a different location or was moved by the end user.

Table 4-12 • Settings for a Dialog Control (cont.)

Setting	Project Type	Description
Left Scrollbar	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Select True to set the scroll bar, if present, at the left side of the dialog. The default value, False, keeps it at the right side. If you want the vertical scrollbar, if present, to be displayed on the left side of the dialog, select True. The default value, False, keeps the scrollbar on the right side.
Minimize	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the end user can minimize this dialog. False means that the option is not present on the title bar.
Modal	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Select True if this dialog is part of the installation wizard and no other dialogs should be placed on top of it.
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Resource Identifier	InstallScript, InstallScript MSI, InstallScript Object	This read-only setting shows the dialog's resource ID—the unique numeric identifier for the dialog.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the caption to the left of the control. Set it to True to align the caption to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Text Style	InstallScript, InstallScript MSI	Select a font to be used for the dialog.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the percentage from the top of the screen where you want the dialog to be placed. A value of 50 centers the dialog vertically. This value is ignored if this dialog is part of an installation wizard, and the previous dialog was in a different location or was moved by the end user.
Track Disk Space	Basic MSI, Merge Module	Select True if this dialog has a control that alerts the end user that a drive is out of space or that checks the value of the OutOfDiskSpace property before performing some action.

Table 4-12 • Settings for a Dialog Control (cont.)

Setting	Project Type	Description
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the dialog is visible, and False means that it is hidden.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the width of the dialog in installer units (1/12 of the height of the system font).

Directory Combo Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

A directory combo is used in conjunction with a directory list and a path edit control to create a browse dialog. The directory combo displays the list of drives mapped on the current system.

When you first draw a directory combo on a dialog, InstallShield prompts you for the name of a Windows Installer property. This property should be the same for the accompanying directory list and path edit. At run time, the installation sets the value of this property based on the end user's selection. For more details, see Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties.

When you select a directory combo control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-13 • Settings for a Directory Combo Control

Setting	Description
Name	Enter a name for this directory combo. The name must be unique among all of the controls in your project.
Base Text Style	This font style is used for the control's text if you specify nothing for the Text Style setting.
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.

Table 4-13 • Settings for a Directory Combo Control (cont.)

Setting	Description
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).
Indirect Property	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Property	Enter the name of a property that is set when an end user end user selects a value from this directory combo. This control populates the first part of the path that is displayed in the directory list, so you must use the same property for the directory combo, the directory list, and path edit when using them together on a browse dialog. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the value of the default selection.
Right-Aligned	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Show CD-ROM	To include CD-ROM volumes in the directory combo, select True.
Show Fixed	To include hard drives in the directory combo, select True.

Table 4-13 • Settings for a Directory Combo Control (cont.)

Setting	Description
Show Floppy	To include floppy drives in the directory combo, select True.
Show RAMDisk	To include RAM volumes in the directory combo, select True.
Show Remote	To include mapped network drives in the directory combo, select True.
Show Removable	To include removable drives in the directory combo, select True.
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	This setting contains the text that is used for the control's text. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Directory List Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

A directory list is used in conjunction with a directory combo and a path edit control to create a browse dialog. The directory list displays the folders below the drive that is selected in the directory combo control, and it populates the value of the path edit control.

When you first draw a directory list on a dialog, InstallShield prompts you for the name of a Windows Installer property. This property should be the same for the accompanying directory combo and path edit. At run time, the installation sets the value of this property based on the end user's selection. For more details, see Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties.

When you select a directory list control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-14 • Settings for a Directory List Control

Setting	Description
Name	Enter a name for this directory list. The name must be unique among all of the controls in your project.
Base Text Style	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).

Table 4-14 • Settings for a Directory List Control (cont.)

Setting	Description
Indirect Property	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Property	Enter the name of a property that is set when an end user selects a value from this directory list. This control populates the path that is displayed in the path edit and changes depending on the selection in the directory combo, so you must use the same property for all three controls when using them together on a browse dialog. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the value of the default selection.
Right-Aligned	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.

Table 4-14 • Settings for a Directory List Control (cont.)

Setting	Description
Text	This setting contains the text that is used for the control's text. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Edit Field Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

An edit field control is a text box in which end users can enter a string or an integer.

When you select an edit field control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.



Project • *(For Basic MSI and Merge Module projects) When you first draw this type of control on a dialog, InstallShield prompts you for the name of a Windows Installer property. InstallShield uses the name that you enter as the value for*

this control's Property setting. At run time, the installation sets the value of this property based on the end user's selection. For more details, see Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties.

Table 4-15 • Settings for an Edit Field Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this edit field. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.

Table 4-15 • Settings for an Edit Field Control (cont.)

Setting	Project Type	Description
Indirect Property	Basic MSI, Merge Module	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Max. Length	Basic MSI, Merge Module	Specify the maximum number of characters that the end user can enter in this control.
MultiLine	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want this control to be multiline control, select True.
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Password	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want this control to behave like a password control, only showing an asterisk (*) in place of each character, select True. If you want this control to behave like a normal edit field control, select False.

Table 4-15 • Settings for an Edit Field Control (cont.)

Setting	Project Type	Description
Property	Basic MSI, Merge Module	Enter the name of a property that is set when an end user enters text in this control. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the value of the default selection.
Property Is Integer	Basic MSI, Merge Module	If the property for this control (specified in the Property setting) has an integer value, select True. If the property's value is a string, select False.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.

Table 4-15 • Settings for an Edit Field Control (cont.)

Setting	Project Type	Description
Text	Basic MSI, Merge Module	This setting contains the text that is used in the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, Merge Module	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Group Box Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A group box can be used to enclose various controls in one area. To move and align the group box along with any individual control that it encloses, hold the CTRL key as you select the group box and each control that you want to select. A group box also provides a label that you can use to express the relationship between the controls that are contained within it.

When you select a group box control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-16 • Settings for a Group Box Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this group box. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.

Table 4-16 • Settings for a Group Box Control (cont.)

Setting	Project Type	Description
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This setting contains the text that is used for the group box label. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.

Table 4-16 • Settings for a Group Box Control (cont.)

Setting	Project Type	Description
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Hyperlink Control

Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

To learn how to add an HTML control to a dialog in an InstallScript project or in an InstallScript MSI project, see Using an HTML Control on a Dialog.

Use the hyperlink control to add a hyperlink to a dialog in your project. The hyperlink control displays an HTML link; clicking the link at run time opens a page in the default browser on the target system.

Important • *Windows Installer 5 and later include support for the hyperlink control. If this control is used on a dialog that is displayed on a system that has an earlier version of Windows Installer, run-time error 2885 occurs, and the installation aborts. Therefore, if you want to use the hyperlink control on a dialog but your installation targets systems that have Windows Installer 4.5 or earlier, it is recommended that you include two versions of the dialog in your project: one with the hyperlink control, and one without it. Add conditions to the dialogs to show or hide them, depending on the version of Windows Installer that is present.*

Following is a sample condition that you could use for the dialog that contains the hyperlink control:

```
VersionMsi >= "5.00"
```

Following is a sample condition that you could use for the alternate dialog that does not contain any hyperlink control:

```
VersionMsi < "5.00"
```

When you select a hyperlink control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-17 • Settings for a Hyperlink Control

Setting	Description
Name	Enter a name for this hyperlink control. The name must be unique among all of the controls in your project.
Base Text Style	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Code Page	If you want the control to use fonts from the code page that is defined in the installation's package, select Database. If you want the control to use fonts from the target system's default code page, select User's System.
Context Help	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Format as Bytes	If you want the installation to attempt to display the value that is entered in the Text setting in bytes, select True. If you select True, the Text setting must contain only a number, without the unit. Then, at run time, it is divided by 512 and displayed as the correct number of kilobytes, megabytes, or gigabytes, depending on the size.
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).

Table 4-17 • Settings for a Hyperlink Control (cont.)

Setting	Description
No Prefix	If the Text setting contains an ampersand that should be displayed as an ampersand character (&), select True. If you are using the ampersand in the Text setting to indicate that the letter that follows the ampersand should be underlined (for example, **&Click** would be displayed as Click), select False.
No Text Wrap	Indicate whether you want to allow the text to wrap in this control. If you select True and the text expands beyond the width of the text area, the end of the string is cut off and replaced with an ellipsis button (...). Otherwise, the text wraps and can extend beyond the height of the text area, if it is not long enough.
Property	Enter the name of a property that is set when an end user clicks this hyperlink. This property can be unique to this control; it does not need to be present in the Property Manager view.
Right-Aligned	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	This setting contains the text that is used for the control. Use HTML markup in this setting for your hyperlink. Following is a sample entry for this setting: **To learn about the latest offerings from ABC Company, visit the ABC Company Web site.** For this sample, the **ABC Company Web site** text is displayed as a hyperlink that points to the **http://www.abccompany.com** Web site. Note that the only protocol that is supported for the hyperlink is HTML. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 4-17 • Settings for a Hyperlink Control (cont.)

Setting	Description
Text Style	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Transparent	If you want the background to show through this control, select True. Note that you may want to avoid making this control transparent if you are placing it on top of a colored bitmap. The text may not be visible if end users change the color scheme of their display. For example, the text in this control may become invisible if an end user sets the high-contrast parameter for accessibility reasons.
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Icon Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The icon control displays a picture of an icon.

When you select an icon control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-18 • Settings for an Icon Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this icon. The name must be unique among all of the controls in your project.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
File Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter the path to and the name of the icon file that you want to use for this control, or click the ellipsis button (...) in this setting to browse to it. InstallShield adds the file to your release at build time. The file must be stored as a binary resource in the installation.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Icon Size	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assuming that your icon file has more than one resource, specify the size of the image that you want to use for the control. The **Use first image** option causes the first image in the file to be displayed and makes it stretch to fit the size of the control, regardless of what you specify for the Stretch to Fit setting.

Table 4-18 • Settings for an Icon Control (cont.)

Setting	Project Type	Description
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Stretch to Fit	Basic MSI, Merge Module	If you want the icon to be resized to fill the area of the control, select True. To have the icon centered if it is smaller than the control or to have the icon cropped if it is bigger than the control, select False.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).

Table 4-18 • Settings for an Icon Control (cont.)

Setting	Project Type	Description
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Line Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The line control creates lines of adjustable length and thickness. You can use lines on a dialog to separate areas of the dialog or add graphical touches.

When you select a line control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-19 • Settings for a Line Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this line. The name must be unique among all of the controls in your project.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.

Table 4-19 • Settings for a Line Control (cont.)

Setting	Project Type	Description
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.

Table 4-19 • Settings for a Line Control (cont.)

Setting	Project Type	Description
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

List Box Control

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The list box control is a standard list box that lets end users select a single option from a list of predetermined options.

When you select a list box control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Project • *(For Basic MSI and Merge Module projects) When you first draw this type of control on a dialog, InstallShield prompts you for the name of a Windows Installer property that identifies all of the items that are displayed in this list box. InstallShield uses the name that you enter as the value for this control's Property setting. At run time, the*

installation sets the value of this property based on the end user's selection. For more details, see Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties.

Table 4-20 • Settings for a List Box Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this list box. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Code Page	Basic MSI, Merge Module	If you want the control to use fonts from the code page that is defined in the installation's package, select Database. If you want the control to use fonts from the target system's default code page, select User's System.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).

Table 4-20 • Settings for a List Box Control (cont.)

Setting	Project Type	Description
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Indirect Property	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Items	Basic MSI, Merge Module	To specify the options that you want to be listed in this control, click the ellipsis button (...) in this setting; this opens the List Items dialog box. To add a new item to the list, click the Add button on the List Items dialog box. For each item, you must enter the text that is displayed in the list box, and a value, which is the value that is assigned to the property if this item is selected.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.

Table 4-20 • Settings for a List Box Control (cont.)

Setting	Project Type	Description
Property	Basic MSI, Merge Module	Enter the name of a property that is set when an end user selects an option from this control. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the text of the default selection.
Property Is Integer	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property for this control (specified in the Property setting) has an integer value, select True. If the property's value is a string, select False.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sorted	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Selecting True causes the items in the list box to be sorted alphabetically. A value of False makes them retain the order that you set for each item in the List Items dialog box.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.

Table 4-20 • Settings for a List Box Control (cont.)

Setting	Project Type	Description
Text	Basic MSI, Merge Module	This setting contains the text that is used for screen readers. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

List View Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

- *Merge Module*

The list view control displays a single column of options; an icon is displayed next to each option.

When you select a list view control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.



Project *• (For Basic MSI and Merge Module projects) When you first draw this type of control on a dialog, InstallShield prompts you for the name of a Windows Installer property that identifies all of the items that are displayed in this list view. InstallShield uses the name that you enter as the value for this control's Property setting. At run time, the installation sets the value of this property based on the end user's selection. For more details, see Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties.*

Table 4-21 • Settings for a List View Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this list view. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.

Table 4-21 • Settings for a List View Control (cont.)

Setting	Project Type	Description
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Icon Size	Basic MSI, Merge Module	Assuming that your icon file has more than one resource, specify the size of the image that you want to use for the control. The **Use first image** option causes the first image in the file to be displayed and makes it stretch to fit the size of the control, regardless of what you specify for the Stretch to Fit setting.
Indirect Property	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Items	Basic MSI, Merge Module	To specify the options that you want to be listed in this control, click the ellipsis button (...) in this setting; this opens the List Items dialog box. To add a new item to the list, click the Add button on the List Items dialog box. For each item, you must enter the text that is displayed in the list view control, and a value, which is the value that is assigned to the property if this item is selected.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).

Table 4-21 • Settings for a List View Control (cont.)

Setting	Project Type	Description
Left Scrollbar	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Property	Basic MSI, Merge Module	Enter the name of a property that is set when an end user selects a value from this control. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the value of the default selection.
Property Is Integer	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property for this control (specified in the Property setting) has an integer value, select True. If the property's value is a string, select False.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sorted	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Selecting True causes the items in the combo box to be sorted alphabetically. A value of False makes them retain the order that you set for each item in the List Items dialog box.

Table 4-21 • Settings for a List View Control (cont.)

Setting	Project Type	Description
Stretch to Fit	Basic MSI, Merge Module	If you want the image to be resized to fill the area of the control, select True. To have the image centered if it is smaller than the control or to have the image cropped if it is bigger than the control, select False.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	Basic MSI, Merge Module	This setting contains the text that is used in the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 4-21 • Settings for a List View Control (cont.)

Setting	Project Type	Description
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Masked Edit Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

The masked edit control is essentially an edit field control that lets end users enter information in a specified format.

When you select a masked edit control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-22 • Settings for a Masked Edit Control

Setting	Description
Name	Enter a name for this masked edit control. The name must be unique among all of the controls in your project.
Base Text Style	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.

Table 4-22 • Settings for a Masked Edit Control (cont.)

Setting	Description
Context Help	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).
Indirect Property	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Mask	Enter the formatting for the mask. To learn more, see MaskedEdit Control in the Windows Installer Help Library.
Max. Length	Specify the maximum number of characters that the end user can enter in this control.
Property	Enter the name of a property that is set with the value that an end user enters in the masked edit control. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the value of the default selection.
Right-to-Left	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.

Table 4-22 • Settings for a Masked Edit Control (cont.)

Setting	Description
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text Style	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Path Edit Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

A path edit control is used in conjunction with a directory combo control and a directory list control to create a browse dialog. The path edit displays the complete path that is assembled from selections that the end user makes in the directory combo and directory list, along with or instead of edits that the end user makes.

When you select a path edit control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-23 • Settings for a Path Edit Control

Setting	Description
Name	Enter a name for this path edit control. The name must be unique among all of the controls in your project.
Base Text Style	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	This property is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).
Indirect Property	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).

Table 4-23 • Settings for a Path Edit Control (cont.)

Setting	Description
Property	Enter the name of a property that is set when an end user enters a value into this path edit or selects one from the directory list. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the value of the default selection.
Right-Aligned	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	This setting contains the text that is used for the control's label. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.

Table 4-23 • Settings for a Path Edit Control (cont.)

Setting	Description
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Progress Bar Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A progress bar is a dynamic, graphical bar that fills up in response to control events.

When you select a progress bar control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-24 • Settings for a Progress Bar Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this progress bar. The name must be unique among all of the controls in your project.

Table 4-24 • Settings for a Progress Bar Control (cont.)

Setting	Project Type	Description
Appearance	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Select the appropriate option: • **Continuous**—The moving progress bar consists of a continuous bar. • **Segmented**—The progress bar consists of a series of rectangles.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control if you specify nothing for the Text Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).

Table 4-24 • Settings for a Progress Bar Control (cont.)

Setting	Project Type	Description
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	Basic MSI, Merge Module	This setting contains the text that is displayed by the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 4-24 • Settings for a Progress Bar Control (cont.)

Setting	Project Type	Description
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Push Button Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A push button control is a button that carries out a command when an end user clicks it.

When you select a push button control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-25 • Settings for a Push Button Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this push button. The name must be unique among all of the controls in your project.

Table 4-25 • Settings for a Push Button Control (cont.)

Setting	Project Type	Description
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting. This setting is enabled if Text is selected for the Control Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Control Style	Basic MSI, Merge Module	Specify whether this control is marked with a text label, an icon, or a bitmap.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
File Name	Basic MSI, Merge Module	Enter the path to and the name of the image file that you want to use for this control, or click the ellipsis button (...) in this setting to browse to it. InstallShield adds the file to your release at build time. The file must be stored as a binary resource in the installation. This setting is enabled if you select Bitmap or Icon for the Control Style setting. This setting is enabled if Bitmap or Icon are selected for the Control Style setting.

Table 4-25 • Settings for a Push Button Control (cont.)

Setting	Project Type	Description
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Icon Size	Basic MSI, Merge Module	Assuming that your icon file has more than one resource, specify the size of the image that you want to use for the control. The **Use first image** option causes the first image in the file to be displayed and makes it stretch to fit the size of the control, regardless of what you specify for the Stretch to Fit setting. This setting is enabled if Bitmap or Icon are selected for the Control Style setting.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Show UAC Shield Icon	Basic MSI, Merge Module	If you want the control to include a User Account Control (UAC) shield icon when end users run the installation on Windows Vista or later systems, select True. The shield icon signals to end users that elevated privileges may be required. If the UAC shield icon should not be included on the control, select False.
Stretch to Fit	Basic MSI, Merge Module	If you want the image to be resized to fill the area of the control, select True. To have the image centered if it is smaller than the control or to have the image cropped if it is bigger than the control, select False. This setting is enabled if Bitmap or Icon are selected for the Control Style setting.

Table 4-25 • Settings for a Push Button Control (cont.)

Setting	Project Type	Description
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This setting contains the text that is used on the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. This setting is enabled if Text is selected for the Control Style setting.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property. This setting is enabled if Text is selected for the Control Style setting.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).

Table 4-25 • Settings for a Push Button Control (cont.)

Setting	Project Type	Description
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Radio Button Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A radio button control is one of two or more mutually exclusive options that end users can select. A radio button must be inserted into a radio button group; it functions as part of that control.

A radio button group and its radio buttons behave as a single control. It is not possible to hide or disable individual buttons within a radio button group. All the buttons in a group must be the same style—for example, either all of them have text or all of them have bitmaps.

Note that when you delete a radio button group, all of its radio buttons are also deleted. In addition, when you change the name of the radio button group control, all of its radio buttons are deleted.

When you select a radio button control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-26 • Settings for a Radio Button Control

Setting	Project Type	Description
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting.

Table 4-26 • Settings for a Radio Button Control (cont.)

Setting	Project Type	Description
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default. The control identifier for the first radio button in a group is the control identifier for the radio button group. The control for each subsequent radio button is the radio button group identifier incremented by one.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Order	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the number that indicates the order in which this radio button is listed in the radio button group. This setting must contain a unique, positive integer, but it does not need to be consecutive with the Order setting of the other radio buttons in this group.
Text	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This setting contains the text that is used for the radio button's label. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 4-26 • Settings for a Radio Button Control (cont.)

Setting	Project Type	Description
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Value	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter the value that you want to be associated with this radio button when end users select it. ***Project*** *• For Basic MSI projects: To make one of the radio buttons selected by default, enter the radio button group's Windows Installer property in the Property Manager view. Type the property in all uppercase letters if you want it to be public in scope.Then, specify the value of the button that you want to appear as the default. For example, if one of your radio buttons has a value of 104 and the radio button group uses the property* ***GRP_PROPERTY1****, enter the following information in the Property Manager view to make this radio button the default selection.* • *Name: GRP_PROPERTY1* • *Value: 104*
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Radio Button Group Control

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Note that in Basic MSI and Merge Module projects, it is not possible to hide or disable individual buttons within a radio button group. All the buttons in a group must be the same style—for example, either all of them have text or all of them have bitmaps.

A radio button group control is a container for radio button controls. A radio button group and its radio buttons behave as a single control. Note that when you delete a radio button group, all of its radio buttons are also deleted. In addition, when you change the name of the radio button group control, all of its radio buttons are deleted.

When you select a radio button group control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.



Project • *(For Basic MSI and Merge Module projects) When you first draw this type of control on a dialog, InstallShield prompts you for the name of a Windows Installer property that identifies all of the buttons that are displayed in this radio button group. InstallShield uses the name that you enter as the value for this control's Property setting. At run time, the installation sets the value of this property based on the end user's selection. For more details, see Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties.*

Table 4-27 • Settings for a Radio Button Group Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this radio button group. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting. This setting has no effect on a radio button group with an image for a label.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.

Table 4-27 • Settings for a Radio Button Group Control (cont.)

Setting	Project Type	Description
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default. The control identifier for the first radio button in a group is the control identifier for the radio button group. The control for each subsequent radio button is the radio button group identifier incremented by one.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Has Border	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To display a border around the radio button group, select True. To omit a border, select False. Use the Sunken setting to further modify the appearance of the border.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Indirect Property	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).

Table 4-27 • Settings for a Radio Button Group Control (cont.)

Setting	Project Type	Description
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Property	Basic MSI, Merge Module	Enter the name of a property that is set when an end user selects one of the radio buttons in this group. This property can be unique to this control; it does not need to be present in the Property Manager view. For information on using this setting to set a default selection in the radio button group, see the radio button's Value setting.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	Basic MSI, Merge Module	This setting contains the text that is used for the radio button group's label. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. This setting has no effect on a radio button group with an image for a label.

Table 4-27 • Settings for a Radio Button Group Control (cont.)

Setting	Project Type	Description
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property. This setting has no effect on a radio button group with an image for a label.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Scrollable Text Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

A scrollable text control displays a long string of text that cannot fit on the dialog; the control includes scroll bars that enable end users to scroll through the text. The LicenseAgreement dialog is an example of a dialog that typically contains a scrollable text control.

When you select a scrollable text control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.



Note • *Unlike the text values for most controls, Windows Installer does not resolve property names or other values within the Scrollable Text control. As a result, the text in the file is displayed exactly as it is written.*

Table 4-28 • Settings for a Scrollable Text Control

Setting	Description
Name	Enter a name for this scrollable text control. The name must be unique among all of the controls in your project.
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
File Name	Enter the path to and the name of the .rtf file that you want to use for this control, or click the ellipsis button (...) in this setting to browse to it. InstallShield adds the file to your release at build time. The file must be stored as a binary resource in the installation.
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Right-Aligned	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.

Table 4-28 • Settings for a Scrollable Text Control (cont.)

Setting	Description
Right-to-Left	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Selection Tree Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A selection tree is a special control that lets end users change the selection state of your features, like in the CustomSetup dialog.

When you select a selection tree control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-29 • Settings for a Selection Tree Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this selection tree. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the control's label if you specify nothing for the Text Style setting.
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).

Table 4-29 • Settings for a Selection Tree Control (cont.)

Setting	Project Type	Description
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.
Indirect Property	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
Property	Basic MSI, Merge Module	Enter the name of a property for the selection tree. The end user can set the property in a browse dialog (a dialog that uses a path edit, directory combo, and directory list control).
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.

Table 4-29 • Settings for a Selection Tree Control (cont.)

Setting	Project Type	Description
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	Basic MSI, Merge Module	This setting contains the text that is used for screen readers. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property. This setting is enabled if Text is selected for the Control Style setting.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 4-29 • Settings for a Selection Tree Control (cont.)

Setting	Project Type	Description
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Text Area Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

A text area control displays text.

When you select a text area control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-30 • Settings for a Text Area Control

Setting	Project Type	Description
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter a name for this text area. The name must be unique among all of the controls in your project.
Base Text Style	Basic MSI, Merge Module	This font style is used for the text if you specify nothing for the Text Style setting.

Table 4-30 • Settings for a Text Area Control (cont.)

Setting	Project Type	Description
Cancel	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Code Page	Basic MSI, Merge Module	If you want the control to use fonts from the code page that is defined in the installation's package, select Database. If you want the control to use fonts from the target system's default code page, select User's System.
Context Help	Basic MSI, Merge Module	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Control Identifier	InstallScript, InstallScript MSI, InstallScript Object	This setting contains a numeric identifier for the control that is unique within the dialog. This identifier is the same as a resource identifier in Visual C++. You should not change the control identifiers for any of the controls that are included with a dialog by default.
Default	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Format as Bytes	Basic MSI, Merge Module	If you want the installation to attempt to display the value that is entered in the Text setting in bytes, select True. If you select True, the Text setting must contain only a number, without the unit. Then, at run time, it is divided by 512 and displayed as the correct number of kilobytes, megabytes, or gigabytes, depending on the size.
Height	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the height of the control in dialog units.

Table 4-30 • Settings for a Text Area Control (cont.)

Setting	Project Type	Description
Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Other Window Styles	InstallScript, InstallScript MSI, InstallScript Object	Click the ellipsis button (...) to display the Other Window Styles dialog box.
No Prefix	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If the Text setting contains an ampersand that should be displayed as an ampersand character (&), select True. If you are using the ampersand in the Text setting to indicate that the letter that follows the ampersand should be underlined (for example, **&Click** would be displayed as Click), select False.
No Text Wrap	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether you want to allow the text to wrap in this control. If you select True and the text expands beyond the width of the text area, the end of the string is cut off and replaced with an ellipsis button (...). Otherwise, the text wraps and can extend beyond the height of the text area, if it is not long enough.
Property	Basic MSI, Merge Module	Enter the name of a property that can supply the initial value for the text. This property can be unique to this control; it does not need to be present in the Property Manager view.
Right-Aligned	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Sunken	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.

Table 4-30 • Settings for a Text Area Control (cont.)

Setting	Project Type	Description
Tab Index	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This setting contains the text that is shown inside the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Basic MSI, Merge Module	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property. This setting is enabled if Text is selected for the Control Style setting.
Tooltip	Basic MSI, Merge Module	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Transparent	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want the background to show through this control, select True. Note that you may want to avoid making this control transparent if you are placing it on top of a colored bitmap. The text may not be visible if end users change the color scheme of their display. For example, the text in this control may become invisible if an end user sets the high-contrast parameter for accessibility reasons.

Table 4-30 • Settings for a Text Area Control (cont.)

Setting	Project Type	Description
Visible	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI and Merge Module projects: Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font). For InstallScript, InstallScript MSI, and InstallScript Object projects: Specify the width of the control in dialog units.

Volume Cost List Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

A volume cost list control displays the disk space requirements that are associated with each volume or drive.

The list has five possible columns. The headings for each column are configurable in the **UIText** table, which you can access in the Direct Editor view. The **UIText** table identifies the values of string entries so that localized strings are displayed at run time when appropriate. Following is a list of the keys in the **UIText** table for each column, along with their default English values:

- VolumeCostVolume—Volume
- VolumeCostSize—Disk Size
- VolumeCostAvailable—Available
- VolumeCostRequired—Required
- VolumeCostDifference—Differences

When you select a volume list cost control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-31 • Settings for a Volume Cost List Control

Setting	Description
Name	Enter a name for this volume cost list. The name must be unique among all of the controls in your project.

Table 4-31 • Settings for a Volume Cost List Control (cont.)

Setting	Description
Available Col Width	Specify the default width in installer units (1/12 of the height of the system font) for the Available column. This column is hidden if the value is the number 0 or it is left blank.
Base Text Style	This font style is used for the contents of the volume cost list if you specify nothing for the Text Style setting.
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Differences Col Width	Specify the default width in installer units (1/12 of the height of the system font) for the Differences column. This column is hidden if the value is the number 0 or it is left blank.
Disk Size Col Width	Specify the default width in installer units (1/12 of the height of the system font) for the Disk Size column. This column is hidden if the value is the number 0 or it is left blank.
Enabled	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Required Col Width	Specify the default width in installer units (1/12 of the height of the system font) for the Required column. This column is hidden if the value is the number 0 or it is left blank.

Table 4-31 • Settings for a Volume Cost List Control (cont.)

Setting	Description
Right-Aligned	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Show CD-ROM	To include CD-ROM volumes in the control, select True.
Show Fixed	To include hard drives in the control, select True.
Show Floppy	To include floppy drives in the control, select True.
Show RAMDisk	To include RAM volumes in the control, select True.
Show Remote	To include mapped network drives in the control, select True.
Show Removable	To include removable drives in the control, select True.
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	This setting contains the text that is used for the control's label. Although you cannot see the label, it is available for screen readers. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property. This setting is enabled if Text is selected for the Control Style setting.

Table 4-31 • Settings for a Volume Cost List Control (cont.)

Setting	Description
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Volume Col Width	Specify the default width in installer units (1/12 of the height of the system font) for the Volume column. This column is hidden if the value is the number 0 or it is left blank.
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Volume Select Combo Control



Project • *This information applies to the following project types:*

- *Basic MSI*
- *Merge Module*

A volume select combo control enables the end user to select from an alphabetical list of volumes, or drives.

When you select a volume select combo control in a dialog of the Dialogs view, InstallShield displays the following settings in the right pane.

Table 4-32 • Settings for a Volume Select Combo Control

Setting	Description
Name	Enter a name for this volume select combo. The name must be unique among all of the controls in your project.
Base Text Style	This font style is used for the control's text if you specify nothing for the Text Style setting.

Table 4-32 • Settings for a Volume Select Combo Control (cont.)

Setting	Description
Cancel	If this is the only control on the dialog that will dismiss the dialog, select True. Clicking the cancel control has the same effect as pressing the ESC key or clicking the Close button on the title bar. The Cancel or Finish button is usually the cancel control. This value is ignored if True is selected for the ErrorDialog setting of the dialog that contains this control.
Context Help	This setting is reserved for future use. Windows Installer does not currently support launching context-sensitive help topics from the installation.
Default	If this is the only control on the dialog that you want to be the default control, which means that it is activated when the end user presses the ENTER key, select True. The Next or OK button is usually the default control.
Enabled	Indicate whether the control is enabled. True means that this control is available (the end user can interact with it). False means that it is not available (grayed out).
Height	Specify the height of the control in Windows Installer user interface units (1/12 of the height of the system font).
Indirect Property	If the property that is associated with this control is referenced indirectly, select True; otherwise, select False. When True is selected for an indirect property is set to True, Windows Installer resolves the referenced property at run time. For example, this check box might use the property **_BROWSE**, whose value is INSTALLDIR. If you select True for the Indirect Property setting, the value of **_BROWSE** will be the current value of the **INSTALLDIR** property. If you select False for the Indirect Property setting, the value of _BROWSE will contain the string INSTALLDIR.
Left	Specify the distance from the left edge of the dialog to the start of the control in installer units (1/12 of the height of the system font).
Left Scrollbar	If you want the arrow and vertical scrollbar to be displayed on the left side of the control, select True. This option should be selected only when True is selected for the Right-to-Left setting.
Property	Enter the name of a property that is set when an end user selects a value from this control. This control populates the first part of the path that is displayed in the directory list, so you must use the same property for the volume select combo, the directory list, and path edit when using them together in a browse dialog. This property can be unique to this control; it does not need to be present in the Property Manager view. To set a default value for this control, make sure the property is a public property by giving it a name that contains only uppercase letters, use the Property Manager view to add the public property, and then assign to it the value of the default selection.

Table 4-32 • Settings for a Volume Select Combo Control (cont.)

Setting	Description
Right-Aligned	The default value of False aligns the text to the left of the control. Set it to True to align the text to the right.
Right-to-Left	For English and other languages that are written from left to right, select False. For Hebrew and those languages that are read from right to left, select True.
Show CD-ROM	To include CD-ROM volumes in the control, select True.
Show Fixed	To include hard drives in the control, select True.
Show Floppy	To include floppy drives in the control, select True.
Show RAMDisk	To include RAM volumes in the control, select True.
Show Remote	To include mapped network drives in the control, select True.
Show Removable	To include removable drives in the control, select True.
Sunken	To give the control's edges a recessed, three-dimensional appearance, select True. To use the default visual style for the control, select False.
Tab Index	Assign an integer that—along with the other controls on this dialog but excluding controls such as static text—specifies the order in which each control on the dialog receives focus when the end user presses the TAB key. The lowest tab index that you can use is the number 0.
Tab Stop	Indicate whether this control receives focus within the tab order. True indicates that the control receives focus; False indicates that the control does not receive focus.
Text	This setting contains the text that is used in the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Text Style	Select the font style, size, and color (if available) in which you want the control's label to be displayed. Leaving the value as <Default> displays the font that is contained in the **DefaultUIFont** property. This setting is enabled if Text is selected for the Control Style setting.

Table 4-32 • Settings for a Volume Select Combo Control (cont.)

Setting	Description
Tooltip	Enter the text that you want to be displayed when the end user places the mouse pointer over the control. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Top	Specify the distance from the top of the dialog to the top of the control in installer units (1/12 of the height of the system font).
Visible	True means that the control is visible, and False means that it is hidden. You can also make the control visible by editing its condition in the Behavior area for the dialog.
Width	Specify the width of the control in Windows Installer user interface units (1/12 of the height of the system font).

Copying and Pasting Controls

In the Dialog Editor, you can copy and paste controls within a dialog or from one dialog to another by using standard Windows keyboard shortcuts or context menus.



Task ***To copy a push button from one dialog to another:***

1. Open the layout for the dialog that contains the button you want to copy.
2. Select the push button and press CTRL+C.
3. Open the second dialog in the Dialog Editor and press CTRL+V to paste the control in the dialog.

The push button is pasted with the same relative size and position as the original. Its other properties are also identical to the original's. The copy of the push button is given a unique name based on the type of control it is. However, you must edit the new control's behavior, since that information is not copied over (Basic MSI projects).



Note • *Radio button groups are a special case. When you copy the group, all of its radio buttons are copied for you. You cannot copy a radio button by itself.*

Cutting and Pasting Controls

In the Dialog Editor, you can move a control from one dialog to another by using standard Windows keyboard shortcuts or context menus.

Task ***To cut a control from one dialog and paste in another:***

1. Right-click the control and click **Cut**, or press CTRL+X. A **Delete Control** dialog box opens to inform you that deleting the control also deletes any associated conditions and actions.
2. Click **Yes** to continue cutting the control from the current dialog.
3. Open the second dialog in the Dialog Editor and press CTRL+V to paste the control in the dialog.

When you are done pasting the control into the new location, specify the control's properties and behavior.

Working with the Wizard Interface

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

This section of the documentation covers aspects of working with the wizard interface of Advanced UI and Suite/Advanced UI projects. The elements of the wizard interface consist of wizard pages and secondary windows (also known as pop-up windows). InstallShield lets you define and customize various styles, brushes, and control themes to make formatting the wizard interface efficient. InstallShield also lets you add, modify, and delete wizard pages and secondary windows, and schedule when each should be displayed.

Elements of the Wizard Interface

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The following two screen shots show the basic elements of the wizard interface.

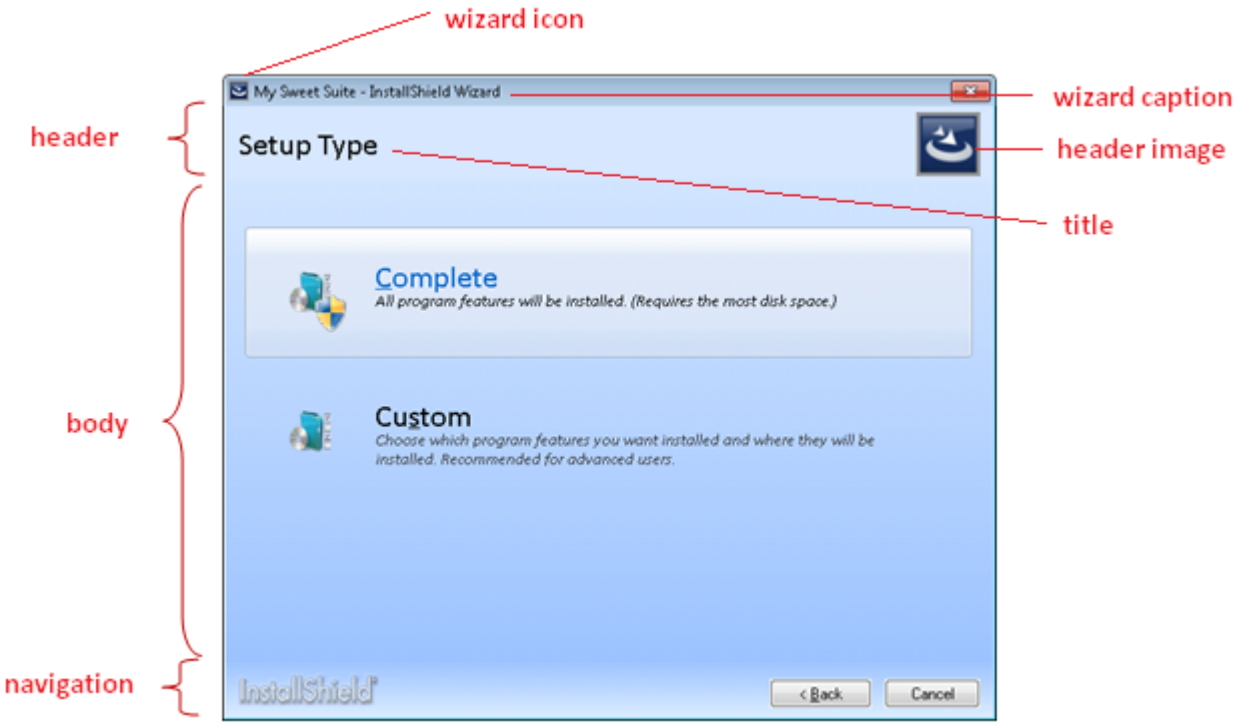


Figure 4-26: Sample Wizard Page in an Advanced UI or Suite/Advanced UI Installation

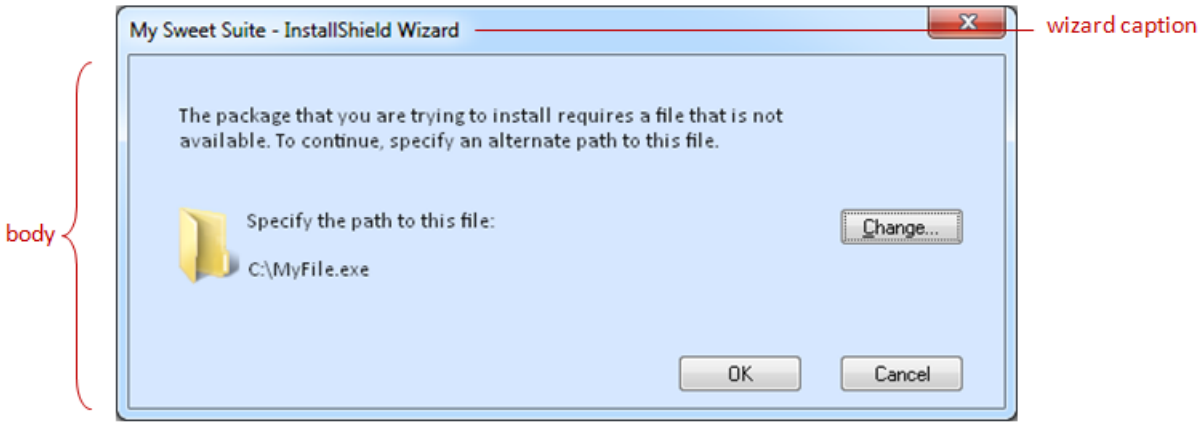


Figure 4-27: Sample Secondary Window in an Advanced UI and Suite/Advanced UI Installation

InstallShield lets you use your own wizard icon, header image, and wizard caption. InstallShield also lets you define and customize styles, brushes, and more for the header, body, and navigation areas of the user interface of an Advanced UI or Suite/Advanced UI project.

Using Styles, Brushes, and Control Themes to Customize the Wizard Interface



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you define and customize various user interface–related styles, brushes, and control themes to easily change the look and feel of the wizard interface without requiring you to manually edit each user interface element individually. Using styles, brushes, and control themes helps to ensure that the wizard interface is formatted consistently throughout the installation.

Styles for the Wizard Interface

InstallShield contains a number of built-in text styles that define text attributes such as color, size, and font name for the text on the wizard UI. In addition, InstallShield contains one built-in font set, which is a collection of fonts (including attributes such as font name, size, and weight). For each font in a font set, you can specify to which language the font is applicable. This enables you to select a different font for each language that your project supports. Font sets work in conjunction with text styles. Text styles reference a font set but can optionally override various font attributes that are defined at the font set level. You can modify any of the built-in text styles and font sets and create your own as needed. To learn more, see:

- Defining a Custom Style for the Wizard Interface
- Applying a Text Style to Text in the Wizard Interface

Brushes for the Wizard Interface

InstallShield contains several built-in brushes. A brush specifies a solid color, a gradient, or an image for various elements of the wizard interface, such as the background of wizard pages and controls. You can modify any of the built-in brushes and create your own as needed. To learn more, see:

- Defining a Custom Brush for the Wizard Interface
- Specifying the Background Brush for the Header, Body, and Navigation Areas of the Wizard Interface
- Overriding the Default Background Brushes for a Wizard Page or Secondary Window

Control Themes for the Wizard Interface

InstallShield enables you to define themes for wizard interface controls. A control theme is a collection of colors for various parts of controls—text color, background color, and border color—and for various states of controls—such as clicked and hovered. You can specify the control theme that you want to use for the controls in your user interface

by default. You can also override the theme for specific controls on wizard pages and windows in your user interface. Control themes give buttons and other items in your installation's user interface a flat look (without the three-dimensional effects) that is reminiscent of the style of Windows Store apps.

The following types of controls support the use of control themes:

- Buttons, including the navigation buttons
- Check boxes
- Radio buttons
- Command links

To learn more, see:

- Defining a Custom Control Theme for the Wizard Interface
- Applying a Theme to a Control in the Wizard Interface

Customizing the Look and Feel of the Wizard Interface

The Wizard Pages node in the Wizard Interface view lets you select default project-wide styles, brushes, and control themes. The Wizard Pages node in this view also lets you override the default values for the wizard caption, wizard icon, and header image, and specify other project-wide UI-related default values.

Use page-specific and control-specific settings such as the Body Background setting that is available for specific wizard pages, secondary windows, and wizard controls if you want to override a specific use of a style. For example, the Body Background setting for the InstallationWelcome wizard page lets you override the project's default background on just the InstallationWelcome wizard page.

Defining a Custom Style for the Wizard Interface



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add custom font sets and text styles to your project and then apply those font sets and text styles to various wizard interface elements. The use of font sets and text styles is helpful for maintaining format consistency and for applying text formatting changes to the wizard interface globally.



Task ***To define a custom style:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, right-click **Styles** and then click **Add Font Set** or **Add Text Style**.

A font set is a collection of fonts (including attributes such as font name, size, and weight). For each font in a font set, you can specify to which language the font is applicable. This enables you to select a different font for each language that your project supports.

Text styles reference a font set but can optionally override various font attributes that are defined at the font set level.

InstallShield adds a a new style under **Styles**. Rename the style as needed.

3. Select the style and then configure its settings in the right pane.

Once you have added a font set, you can select it for the Font List setting of any of the text styles that are defined in the Wizard Interface view. If you later want to change the list of fonts that are used for text styles that are based on that font set, select the font set and edit its settings.

Once you have added a text style, you can select it for various style-related settings in the Wizard Interface view. If you later want to change the appearance of the wizard interface elements that use that style, select the style and edit the values for its settings.

Changes that you make to a style affect all of the wizard interface elements that use that style.

Applying a Text Style to Text in the Wizard Interface



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to apply text styles to various wizard interface elements. If you later want to change the appearance of the wizard interface elements that use that text style, you can easily make changes to the text style, rather than separately editing each element that uses the text style.

When you are applying a text style to text, you can apply it at the global level, or you can apply it at the individual wizard page control level.



Task ***To apply a text style to text globally:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, click **Wizard Pages**. InstallShield displays the global wizard interface settings in the right pane.
3. In the following settings, select the appropriate text styles:
   - Header Text Style
   - Navigation Text Style

InstallShield changes the text style of the applicable UI elements throughout the project.

Task ***To apply a text style to specific text:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, expand the **Wizard Pages** node or the **Secondary Windows** node, and find the wizard page or window that contains the text to which you want to apply a style.
3. Select the control that contains the text to which you want to apply a style.
4. In the right pane, expand the **Text Style** setting, and use its subsettings to specify the appropriate text styles. For more information, see the inline help that is displayed in the bottom right pane when you select these settings.

InstallShield changes the text style of the selected control.

Defining a Custom Brush for the Wizard Interface

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

A brush specifies a solid color, a gradient, or an image for various elements of the wizard interface, such as the background of wizard pages and controls. InstallShield lets you add custom brushes to your project and then apply those brushes to various wizard interface elements.

Task ***To define a custom brush:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, right-click **Brushes** and then click **Add Brush**. InstallShield adds a a new brush under **Brushes**. Rename the brush as needed.
3. Select the brush and then configure its settings in the right pane.

Once you have added a brush, you can select it for various background-related settings in the Wizard Interface view. For example, to specify the brush that you want to use by default for the background of the navigation area of the wizard pages, use the Navigation Background setting.

Specifying the Background Brush for the Header, Body, and Navigation Areas of the Wizard Interface

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

By default, InstallShield enables you to use different brushes for the header, body, and navigation areas of your wizard pages. In some cases, though, you may want to use only one brush (for example, a single background image or a vertical gradient) that spans all three of those areas. The following instructions explain how to customize using three different brushes or one brush.

Tip *• For information on defining brushes, see Defining a Custom Brush for the Wizard Interface.*

Using Three Different Brushes for the Header, Body, and Navigation Areas

Task

To use three different brushes for the header, body, and navigation areas:

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, click **Wizard Pages**. The global wizard interface settings are available in the right pane.
3. In the **Default Body Background** setting, select the brush that you want to use as the background for the body area of the wizard interface.
4. In the **Header Background** setting, select the brush that you want to use as the background for the header area of the wizard interface.
5. In the **Navigation Background** setting, select the brush that you want to use as the background for the navigation area of the wizard interface.

Important *• Note that if you select a brush in the aforementioned background settings, you should delete any value in the Full Wizard Background setting. The wizard interface supports use of a single brush for the full wizard background, or separate brushes for the header, body, and navigation areas; it does not support specification of brushes for all four of those areas simultaneously.*

Using One Brush that Spans the Header, Body, and Navigation Areas

Task ***To use a single brush for the header, body, and navigation areas:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, click **Wizard Pages**. The global wizard interface settings are available in the right pane.
3. In the **Full Wizard Background** setting, select the brush that you want to use as the background for the navigation, body, and navigation areas of the wizard interface.

Important • *Note that if you select a brush in the aforementioned setting, you should delete any values from the other background settings: Default Body Background, Header Background, and Navigation Background. The wizard interface supports use of a single brush for the full wizard background, or separate brushes for the header, body, and navigation areas; it does not support specification of brushes for all four of those areas simultaneously.*

Overriding the Default Background Brushes for a Wizard Page or Secondary Window

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

In most cases, each wizard page and secondary window in the user interface of an Advanced UI or Suite/Advanced UI project has the same background. In some cases, you may want to override the default background for a specific wizard page or secondary window in your project. You can do so by selecting a different brush for the background of that wizard page or window.

Task ***To apply a brush to a specific wizard page or secondary window:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, expand the **Wizard Pages** node or the **Secondary Windows** node, and then click the wizard page or window to which you want to apply a brush. InstallShield displays the settings for the selected wizard page or window in the right pane.
3. In the **Body Background** setting, select the brush that you want to use for the selected wizard page or secondary window.

InstallShield changes the background of the selected wizard page or secondary window.

Defining a Custom Control Theme for the Wizard Interface



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to define themes for wizard interface controls. A control theme is a collection of colors for various parts of controls—text color, background color, and border color—and for various states of controls—such as clicked and hovered. You can specify the control theme that you want to use for the controls in your user interface by default; this is also the theme that is used for all of the navigation buttons (for example, Back buttons and Next buttons) in your user interface. You can also override the theme for specific controls on wizard pages and windows in your user interface. Control themes give buttons and other items in your installation's user interface a flat look (without the three-dimensional effects) that is reminiscent of the style of Windows Store apps.

The following types of controls support the use of control themes:

- Buttons, including the navigation buttons
- Check boxes
- Radio buttons
- Command links



Task ***To define a control theme:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, right-click **Control Themes** and then click **Add Control Theme**. InstallShield adds a a new control theme under **Control Themes**. Rename the control theme as needed.
3. Select the control theme and then configure its settings in the right pane.

Once you have added a control theme, you can select it for various theme-related settings in the Wizard Interface view. If you later want to change the appearance of controls that are based on that control theme, select the control theme and edit its settings.

Changes that you make to a control theme affect all of the wizard interface elements that use that control theme.

Applying a Theme to a Control in the Wizard Interface



Project *• This information applies to the following project types:*

- *Advanced UI*

- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to apply control themes to buttons, check boxes, radio buttons, and command links. If you later want to change the appearance of the wizard interface elements that use a control theme, you can easily make changes to the control theme, rather than separately editing each element that uses the control theme.

You can apply a control theme to a few individual controls and leave the remaining ones alone, or you can specify a global theme for your project and override it for a few individual controls. Changing the theme at the global level does not change the theme of the controls that have a per-control theme specified.

Note that you can apply control themes to navigation buttons at the global project level, but not at the individual control level.



Task ***To apply a control theme to controls globally:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, click **Wizard Pages**. InstallShield displays the global wizard interface settings in the right pane.
3. In the **Default Control Theme** setting, select the appropriate theme that you want to use by default for controls such as buttons (including navigation buttons), check boxes, radio buttons, and command links.

InstallShield changes the theme of all of the controls that do not have a per-control theme selected.



***Tip** • If you want the installation to use user-chosen system colors for controls instead of the colors that are defined for a particular control theme, select the **(No Theme)** option in the Default Control Theme setting.*



Task ***To apply a theme to a control, overriding the default theme selection:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, expand the **Wizard Pages** node or the **Secondary Windows** node, and find the wizard page or window that contains the control (button, check box, radio button, or command link) to which you want to apply a theme.
3. Select the control to which you want to apply a theme.
4. In the right pane, find the **Control Theme** setting, and select the theme that you want to use for the selected control.

InstallShield changes the theme of the selected control.

Tip *• To avoid overriding the default control theme for a selected control, select the (No Theme) option in the Control Theme setting, or leave this setting blank.*

Selecting the Format for the Wizard Interface

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you select from several different wizard formats for your installation's wizard interface:

- **Glass**—The installation uses user-chosen system colors to display the caption bar, header, and navigation areas of the wizard interface. It also uses the glass effect (translucency) that was introduced in Windows Vista for these same areas of the wizard interface. Note that with this format, the installation ignores any brushes that you have defined for the caption bar, header, and navigation areas of your wizard interface.

  Note that some operating systems do not fully support this format. Thus, in some cases, the installation may use the Wizard 97 format instead of the Glass format.

  This is the default option.

- **Aero**—This format typically has the same look and feel as the Glass format.

  Note that some operating systems do not fully support this format. Thus, in some cases, the installation may use the Wizard 97 format instead of the Aero format.

- **Wizard 97**—This format is the traditional wizard format. The installation uses the brushes that you have defined for the various areas of your wizard interface.

- **Wizard Lite**—This format is similar to the Wizard 97 format, except that it does not include a header (the area that shows the title, plus sometimes an image near the upper corner of the wizard page). The installation uses the brushes that you have defined for the other areas of your wizard interface.

Note that in some scenarios, an installation uses the Wizard 97 format instead of the Glass or Aero formats:

- Windows 8 and Windows Server 2012 systems do not have translucency support. On these systems, a Glass-formatted wizard interface uses the Wizard 97 format. However, an Aero-formatted wizard interface uses Aero, but without the glass effect.

- Windows 7, Windows Vista, Windows Server 2008 R2, and Windows Server 2008 have translucency support that end users can enable or disable. On these systems, Glass-formatted wizard interfaces and Aero-formatted wizard interfaces use the Wizard 97 format if translucency is disabled. These wizard interfaces also use the Wizard 97 format if the desktop composition feature on the target system is disabled.

- On Windows XP and Windows Server 2003, Glass-formatted wizard interfaces and Aero-formatted wizard interfaces use the Wizard 97 format.

Note that the wizard interface editor in the Wizard Interface view in InstallShield does not use the Glass format or the Aero format to show editable previews of the interface. If you select either of these formats for your Advanced UI or Suite/Advanced UI project, the wizard interface editor uses the Wizard 97 format to show editable previews.



Task ***To select the format that you want to use for wizard pages in your wizard interface:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, click **Wizard Pages**.
3. In the right pane, find the **Wizard Format** setting and select the appropriate option.

Creating a New Blank Wizard Page in an Advanced UI or Suite/Advanced UI Project



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to add a new blank wizard page to your project.



Task ***To create a new blank wizard page in an Advanced UI or Suite/Advanced UI project:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, right-click **Wizard Pages** and then click **Add Blank Page**.

InstallShield adds a a new wizard page under **Wizard Pages**. Rename the wizard page as needed. Configure its settings in the right pane, and use the center pane to preview it and modify its layout.

Adding a Predefined Wizard Page in an Advanced UI or Suite/Advanced UI Project



Project *• This information applies to the following project types:*

- *Advanced UI*

- *Suite/Advanced UI*

***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to add various predefined wizard pages to your Advanced UI or Suite/Advanced UI project. Examples include:

- A page that allows end users choose the installation directory for an .msi package
- A page that lets end users enter customer information and serial numbers
- A page that shows a feature tree, as well as feature descriptions and sizes, and allows end users to select which features to install
- A database login page that allows end users to enter database server login information (database server name, authentication credentials, database catalog name, etc.) in order to establish a connection to the database server that is targeted by one or more .msi packages in the suite

Task ***To add a predefined wizard page in an Advanced UI or Suite/Advanced UI project:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, right-click **Wizard Pages** and then click **Add Predefined Page**. The **User Interface Wizard** opens.
3. Complete the panels in the wizard as needed.

InstallShield adds the new predefined page to your project. Configure its settings in the right pane, and use the center pane to preview it and modify its layout.

***Note** • When you add the browse-for-installation-folder type of wizard page (called BrowseFolder) to your project, InstallShield adds a placeholder image control to it so that you can display a lock image on the page. Ensure that you either configure the control's Resource setting to indicate the file that you want to display with this control, or, to exclude an image, delete the image control.*

Creating a New Blank Secondary Window in an Advanced UI or Suite/Advanced UI Project

***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to add a new blank secondary window to your project.

Task ***To create a new blank secondary window in an Advanced UI or Suite/Advanced UI project:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, right-click **Secondary Windows** and then click **Add Blank Window**.

InstallShield adds a new window at the end of the list of secondary windows. Rename the window as needed. Configure its settings in the right pane, and use the center pane to preview it and modify its layout.

Sequencing Wizard Pages

***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Use the Wizard Interface view to indicate the order in which your Advanced UI or Suite/Advanced UI installation should display the wizard pages at run time. The wizard pages are listed under the Wizard Pages node in this view in the order that the installation displays them at run time.

Task ***To resequence a wizard page in your project:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, under the **Wizard Pages** node, right-click the wizard page whose order you want to change, and then click **Move Up** or **Move Down**.

InstallShield moves the wizard page up or down in the list.

Editing the Layout and Behavior of a Wizard Page or Secondary Window

***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Use the Wizard Interface view to edit the layout and behavior of wizard pages and secondary windows in your installation.

Adding a Control to a Wizard Page or Secondary Window



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

If you select a wizard page or secondary window in the Wizard Interface view, the toolbar that InstallShield shows directly above the wizard interface preview pane includes several different buttons and other controls that let you add new controls.



Task ***To add a control to the wizard interface:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, click the wizard page or secondary window that you want to modify. The wizard editor in the center pane shows the page or secondary window.
3. In the toolbar above the wizard interface preview pane, click the appropriate New Control button (the Label, Text Box, Button, or **Combo Box** button), or select one of the control types that are listed in the drop-down list next to the control buttons. For information about each of the available types of controls, see Wizard Interface View Toolbar.

   InstallShield adds the control near the upper-left corner of the body area of the page or secondary window. In addition, InstallShield adds a subnode for the control under the selected wizard page or window.
4. Place the cursor in the middle of the control, and then drag it to move it to the appropriate location.
5. In the right pane, configure the control's settings.



***Tip** • The order in which the controls are listed under the a wizard page's node matches the tab order.*

If you are using more than one group of radio button controls on a page, ensure that the Style setting includes WS_GROUP for the first radio button of each group.

Changing the Tab Order of Controls on a Wizard Page or Secondary Window



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

In most cases, end users can use the TAB key to move the focus from one control on a wizard page or window to another. By default, the tab order is the order in which the controls were added to the wizard page or window.



Task ***To change the tab order of the controls on a wizard page or window:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, expand the node of the wizard page or secondary window whose tab order you want to modify.

   Each control on the page or window has its own subnode. The controls are listed in tab order. That is, the first control under a page node is the first control that receives focus on that page.
3. Right-click the control that you want to resequence in the tab order, and then click **Move Up** or **Move Down**.
4. Continue moving the controls as needed until the order that the controls are listed under the page reflects the appropriate tab order.

Note that in some cases, a control may not be included in the tab order. For example, a disabled control would not receive focus as an end user tabs from one control to another control.

Designating Keyboard Shortcuts and Using Ampersands (&) in the Wizard Interface



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

In wizard interface strings, an ampersand (&) is a special character that sometimes designates a keyboard shortcut, but other times indicates that a literal ampersand should be displayed. For button, check box, and radio button controls, a single ampersand always designates the keyboard shortcut that is used to click or select the control; a double ampersand (&&) indicates that a single ampersand should be displayed. For label controls, the interpretation of an ampersand depends on the style of the label control in which the ampersand is used.

Table 4-33 • Using Ampersands in Label Controls of the Wizard Interface

Sample String Entry for a Label Control	Status of SS_NOPREFIX in the Style Setting of the Control	String That Is Displayed in the Wizard Interface	Notes
Enter your &serial number for New && Improved Product	SS_NOPREFIX = False	Enter your serial number for New & Improved Product	When an end user presses the ALT key, the "s" in the string is underlined. The keyboard shortcut of ALT+s moves the focus to the control that appears directly after this label in the tab order. The double ampersand between "New" and "Improved" indicates that a single ampersand should be displayed.
Enter your serial number for New & Improved Product	SS_NOPREFIX = True	Enter your serial number for New & Improved Product	The True value for the SS_NOPREFIX constant prevents any ampersands in the control's string from being used to designate a keyboard shortcut.

By default, when you add a new label control to your wizard interface, the SS_NOPREFIX subsetting under the label control's Style setting is set to False. You can change this value as needed.

If your wizard interface includes a label control that should never contain a keyboard shortcut, consider selecting True for the SS_NOPREFIX subsetting of that control. This ensures that any ampersands that are included in the string entries for such controls are not inadvertently interpreted as keyboard shortcuts.

If your product name includes an ampersand, you may want to use two separate product name properties in your wizard interface—one property whose value uses a single ampersand, and one property whose value uses the escaped ampersand (a double ampersand). Depending on the type and style of each control that contains the product name, you would use the appropriate product name property as needed—for example:

Table 4-34 • Sample Usage of Two Different Properties for a String that Includes an Ampersand

Property	Property Value	Examples of Uses for This Property
[ProductName]	New & Improved Product	• Label controls in which SS_NOPREFIX = True • Wizard Caption setting on the Wizard Pages node—This value is used as the caption in the caption bar of wizard pages. The default value of this setting is: [ProductName] - InstallShield Wizard • Title setting for a wizard page—The title text is displayed in the header area of wizard pages.

Table 4-34 • Sample Usage of Two Different Properties for a String that Includes an Ampersand (cont.)

Property	Property Value	Examples of Uses for This Property
[EscapedProductName]	New && Improved Product	• Label controls in which SS_NOPREFIX = False • Controls such as check boxes, command links, and radio buttons, which do not support the SS_NOPREFIX style setting and always interpret single ampersands as being used to designate keyboard shortcuts

Using Navigation Buttons on Wizard Pages



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Each wizard page in an Advanced UI or Suite/Advanced UI project can have one or more of the following built-in navigation buttons:

Table 4-35 • Built-In Navigation Buttons for Wizard Pages

Button	Description of Default Behavior
Next	Move to the next wizard page in the interface, skipping any pages whose Visible conditions evaluate to false. Clicking this button has no effect if the current wizard page is the last page in the interface. The Next navigation button behaves similarly to controls that use a Set Active Page action, which triggers a move to a specific page.
Back	Move to the previous wizard page in the interface, skipping any pages whose Visible conditions evaluate to false. Clicking this button has no effect if the current wizard page is the first page in the interface. The Back navigation button behaves similarly to controls that use a Set Active Page action, which triggers a move to a specific page.
Cancel	Display a message box with question IDS_SUITE_CONFIRMCANCEL; on Yes, cancel the wizard and move to the last wizard page. In most cases, the last wizard page should not show a Cancel button, since cancelling the installation at that point does not have any effect. If a Cancel button is displayed, clicking it would result in closing the wizard.

Table 4-35 • Built-In Navigation Buttons for Wizard Pages (cont.)

Button	Description of Default Behavior
Install	Start the installation and move to the next wizard page. The Install navigation button behaves similarly to controls that use an Install action, which start the installation and then move to a specific wizard page.
Finish	After a successful installation or maintenance, if a restart is required, ask whether to restart (message IDS_SUITE_ASKREBOOT), and then trigger the appropriate response. In either case (required or not required, immediate restart or delayed restart), close the wizard.

Including a Navigation Button on or Excluding It from a Wizard Page

When you select a wizard page in the Wizard Interface view of an Advanced UI or Suite/Advanced UI project, the grid on the right side of the view shows the settings for the selected page. The Navigation area settings let you configure the navigation buttons for the page.

Task ***To include or exclude a navigation button:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, click the wizard page whose buttons you want to modify.
3. In the **Navigation** area of the grid on the right side of the view, do the following:
   - To include a specific button on the wizard page, select **Yes** in that button's setting, and then configure the subsettings under that button's setting as needed.
   - To exclude a specific button from a wizard page, select **No** in that button's setting, and then configure the subsettings under that button's setting as needed.

To learn about changing the default behavior of navigation buttons, see Overriding the Default Behavior of a Navigation Button on a Wizard Page.

Overriding the Default Behavior of a Navigation Button on a Wizard Page

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

If the Click event of a navigation button includes either a Set Active Page action or an Install action, and if that action moves through the wizard interface, this prevents the default behavior described above. The default behavior is prevented even if the scheduled action is skipped because of a false condition for the action.

For example, you may want to override the default behavior for the Install button: instead of moving to the next wizard page in the interface, you may want the installation to proceed to a specific wizard page. In this scenario, you must add an Install action to the Install button's Click setting, and specify the specific subsequent page. If you then add a condition to the action, and the condition prevents the Install action from running, the default behavior for the Install button is suppressed. That is, the installation does not start when the end user clicks the Install button. (Note that in this case, you may want to define a condition for the navigation button's Enabled setting to ensure that the Install button is disabled if appropriate. Otherwise, end users may become confused when they are able to click an Install button that has no effect.)

Note that extension DLLs actions cannot prevent the default action except by actually calling the corresponding methods on the ISuiteUIExtension interface. The Advanced UI or Suite/Advanced UI engine cannot determine whether the extension chose to call the method. If you need an extension action to override the default behavior, but only navigate or install under certain conditions, add a Set Active Page or Install action with a false condition such as the empty None group.

Populating Combo Box and List Box Controls in the Wizard Interface



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add to the wizard interface one or more combo box controls and list box controls with predefined selectable options. The process involves configuring the control's settings to define the content for the selectable options. The process also involves the use of two properties: one that defines the list of options that you want to display in the control, and one that the installation uses to store the option that the end user selects at run time.



Task ***To populate a combo box control or a list box control with predefined options:***

1. Associate the control with two properties: one that defines the list of options that you want to display in the control, and one that the installation uses to store the option that the end user selects at run time:
   a. In the View List under **User Interface**, click **Wizard Interface**.
   b. In the **Wizard Interface** explorer, expand the wizard page or secondary window that you want to modify, and then select the combo box control or list box control that you want to configure.
   c. In the **Property** setting, enter the name of the Advanced UI or Suite/Advanced UI property—for example, **SELECTIONPROPERTY**—that you want to associate with the control. At run time, the installation uses this property to store the value that the end user selects.
   d. In the **Content Property** setting, enter the name of the Advanced UI or Suite/Advanced UI property that you want to use to identify the options that you want to list in this control—for example, **CONTROLOPTIONS**.

2. Define the list of options that you want to display in the combo box control or the list box control:

   a. In the **Content Property** setting, click the ellipsis button (**...**). The **Edit List Options** dialog box opens.

   b. In the **Text** column, enter one of the options that you want to display in the control.

      When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

   c. In the **Value** column, enter the value that you want to associate with the list option text that you entered in step 2b. This is the property value that is stored in **SELECTIONPROPERTY**—the property that you entered in step 1c—if the end user selects the list option that you entered in the **Text** column.

   d. To add additional text-value pairs, click the **New** button, and then complete the fields in the row that InstallShield adds.

      For example, you might enter the following data:

Text	Value
ID_Option1	Value1
ID_Option2	Value2
ID_Option3	Value3

   e. Change the order of the list options if appropriate.

      The installation displays the list options in the order that you list them on this dialog box. To change the order of the list options, select a row, and then click the Up or Down buttons as needed. To select multiple consecutive rows, click the first row, press and hold Shift, and then click the last row. To select multiple non-consecutive rows, click one of the rows, press and hold down CTRL, and then click each additional row.

3. Click **OK**.

Specifying the Default Option for the Combo Box Control or the List Box Control

If you want to specify the default option for a combo box control or a list box control, use the Property Manager view to set the value of the Advanced UI or Suite/Advanced UI property equal to the property value that corresponds with the default selectable option. Thus, if you want *ID_Option3* to be selected by default, add the **SELECTIONPROPERTY** property to the Property Manager view, and set its value to *Value3*. To leave the control blank by default, do not define the property in the Property Manager view.

Working with End-User Data from a Combo Box Control or a List Box Control

At run time, end users can make a single selection from the combo box or the list box; this sets the control's property (which is **SELECTIONPROPERTY** in the aforementioned procedure) to the value that is associated with the selected option—for example, *Value1*, *Value2*, or *Value3*. You can trigger different types of behavior based on the value of the **SELECTIONPROPERTY** property. Examples are:

- You can enable or disable a control on the wizard page or secondary window. To do this, use the control's Enabled setting to define a Property type of condition and tie the enabled state of the control to a particular value of the property.

- You can display or hide a particular control on a wizard page or secondary window. To do this, use the control's Visible setting to define a Property type of condition and tie the enabled state of the control to a particular value of the property.
- You can display or skip a particular wizard page or secondary window. To do this, use the wizard page's or secondary window's Visible setting to define a Property type of condition and tie the enabled state of the wizard interface to a particular value of the property.
- You can use the value of the Advanced UI or Suite/Advanced UI property to set a particular property in a package in the Advanced UI or Suite/Advanced UI installation. To do this, use the command line and silent command line settings that are available as subsettings in the Operation area of the grid in the Packages view. For more information, see Using Advanced UI and Suite/Advanced UI Formatted Expressions to Dynamically Configure Command Lines.

Configuring Validation for a Control on a Wizard Page or Window



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you implement validation for some of the various wizard interface controls. For example:

- You can ensure that end users enter a serial number in a text box in a specified format.
- You can configure validation for a text box control to check whether the folder an end user specifies in the text box is valid.
- You can prevent end users from selecting certain combinations of check boxes that do not work together.

Validation for a control checks the property that is associated with that control. When an end user clicks a check box, for example, the associated property is potentially changed to one of two new values. Validation offers a method for checking whether the new value is appropriate.

Various types of controls trigger validation at different events:

Table 4-36 • Triggers for Different Types of Validation

Types of Controls	Event that Triggers Validation
Text box, password box	Validation occurs when the entry in the control changes.

Table 4-36 • Triggers for Different Types of Validation (cont.)

Types of Controls	Event that Triggers Validation
Button, command link, check box, radio button, image button	Validation occurs when the end user clicks the control.
Combo box, list box, checked list box, feature selection tree	Validation occurs when the selection for the control changes.

You can use the subsettings under the Text Style setting to specify different text styles for the control to give end users a visual indication when the control is in various states (default, valid, or invalid).

You can also use the subsettings in the Events area to trigger an action when the end user clicks or uses the control. If you have actions and validation configured for a control, the validation runs before the actions. For more information about actions, see Configuring an Action for an Element in the Wizard Interface.



Task ***To configure validation for a control:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, select the wizard page or secondary window that you want to modify.
3. In the wizard editor, select the control that requires validation.
4. In the **Validate** setting, click the **Validation** button and then point to the type of validation that you want to occur.

   One of the types of validation lets you call a function in a DLL file that you have created. If your project includes any DLL files in the Support Files view, InstallShield makes them available in this list, and lets you overwrite the validation function name with the function in the DLL that you want to call.

The following table describes the types of validation that are available. Note that some types of validation are applicable to only certain types of controls. For example, the value length validation is applicable only to the text box and password box controls.

Table 4-37 • Types of Validation for Wizard Interface Controls

Type of Validation	Description
Existing File	This type of validation checks whether the value of the property that is associated with this control contains the fully qualified path and name of a file that is present. The end user typically sets the value of the property when entering the path and file name in this control.

Table 4-37 • Types of Validation for Wizard Interface Controls (cont.)

Type of Validation	Description
Existing Folder	This type of validation checks whether the value of the property that is associated with this control contains the fully qualified path and name of a folder that is present. The end user typically sets the value of the property when entering the path and folder name in this control.
New File in Existing Folder	This type of validation checks whether the value of the property that is associated with this control contains the fully qualified path and name of a file that is not present. The end user typically sets the value of the property when entering the path and file name in this control. Note that the path for the file must exist for this type of validation.
Value Length	This type of validation checks whether the value of the property that is associated with this control contains a certain number of characters. The end user typically sets the value of the property when entering text in this control. To configure this type of validation, specify the minimum and maximum values in the subsettings under this setting.
Value Mask	This type of validation checks whether the value of the property that is associated with this control matches a particular format. The end user typically sets the value of the property when entering information such as a serial number in this control. To configure this type of validation, specify the required format in the Pattern setting under this setting. Enter any of the following characters in place of a digit: **# % @** Enter any of the following characters in place of a character: **& ^ ? '** Any other characters that you enter must match exactly.
Browse for DLL Action	This type of validation calls a function in a DLL file that you have created and added to your project through the Support Files view. For example, you can use a DLL file to check the validity of the property's value—that is, the input that the end user enters for the control.

Configuring an Action for an Element in the Wizard Interface



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you define actions that are launched when a wizard page or secondary window is opened or closed. This capability enables your installation to perform any initialization such as dynamically retrieving values for a combo box. It also lets you schedule actions to occur before the wizard interface is displayed at run time.

In addition, InstallShield lets you define actions that an end user triggers when using some of the various wizard interface controls. For example, you can add a Print button to a LicenseAgreement wizard page, and define a print action for that button control; when an end user clicks the Print button, the Print dialog box opens, enabling the end user to print the license agreement.

If you have actions and validation configured for a control, the validation runs before the actions. The actions run for both validation success and validation failure. They also run when the splash page is dismissed, when an image button gets or loses focus, or when an end user scrolls to the bottom of a rich text box. For more information about validation, see Configuring Validation for a Control on a Wizard Page or Window.

Task ***To configure an action for a wizard page, secondary window, or control:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, select the wizard page or secondary window that you want to modify.
3. Do one of the following if you are configuring an action for a control:
   - To configure the action of one of the navigation controls (the Next, Back, Cancel, Install, or Finish buttons)—In the right pane, expand the settings for the control whose action you want to modify.
   - To configure the action of any of the other controls—In the wizard editor, select the control whose action you want to modify.
4. Do one of the following:
   - To schedule an action to run when the selected wizard page or window opens: In the **Page Entered** setting, point to the **New Action** button, and then click the type of action that you want to configure.
   - To schedule an action to run when the selected wizard page or window closes: In the **Page Exited** setting, point to the **New Action** button, and then click the type of action that you want to configure.
   - To schedule an action for a control: In the **Click** setting, or in an action-related setting under the **Events** setting, point to the **New Action** button, and then click the type of action that you want to configure.
5. Configure the subsettings under the action as needed.

If you enter more than one action statement, the actions are run in the order in which they are listed in the action setting.

One of the types of actions lets you call a function in a DLL file that you have created. If your project includes any DLL files in the Support Files view, InstallShield makes them available in this list, and lets you overwrite the action function name with the function in the DLL that you want to call.

The following table describes the types of actions that are available.

Table 4-38 • Control Actions

Type of Action	Description
Set Property	This type of action sets a specific property to a specific value.
Install	This type of action starts the installation and moves to a specific wizard page.
Open	This type of action opens a file or Web page.
Print	This type of action launches the Print dialog box, enabling an end user to print the specified file.
Browse for Folder	This type of action launches the Browse for Folder dialog box when an end user clicks the control. When the end user selects a folder in this dialog box and clicks the OK button, the dialog box closes, and the installation sets the value of a specific property to the full path of the folder that the end user selected. To configure this type of action, configure the subsettings under this action.
Browse for File	This type of action launches the file browse dialog box when an end user clicks the control. When the end user selects a file in this dialog box and clicks the Open button, the dialog box closes, and the installation sets the value of a specific property to the full path and file name of the file that the end user selected. To configure this type of action, configure the subsettings under this action.
Browse for New File	This type of action launches the new-file browse dialog box when an end user clicks the control. The dialog box lets the end user browse to a location where a new file should be saved. When the end user specifies a file in this dialog box and clicks the Save button, the dialog box closes, and the installation sets the value of the property that you have specify in the Property subsetting to the full path and file name of the file that the end user specifies. To configure this type of action, configure the subsettings under this action.
Set Active Page	This type of action moves to a specific wizard page.
Show Window	This type of action shows a secondary window as a modal window.

Table 4-38 • Control Actions (cont.)

Type of Action	Description
Close Window	This type of action closes a main wizard page or secondary window, or in some cases, provides conditional closing of a secondary window. Close Window accepts parameters corresponding to the following predefined Return Code IDs: IDOK, IDCANCEL, IDABORT, IDRETRY, IDIGNORE, IDYES, IDNO, and IDCLOSE. The behavior of the Close Window action differs slightly on wizard pages and secondary windows, as noted following: • For wizard pages, the Close Window action prompts the end user to cancel if its Return Code parameter is set to IDCANCEL (and then interrupts the wizard if the end user specifies Yes). For all other Return Code IDs, the wizard immediately closes. • For secondary windows, the Close Window action closes the secondary window and in special cases such as with the ISRMFilesInUse and ISRMFileInUse secondary windows, the specified Return Code value is returned. InstallShield currently provides the following secondary windows that contain customized behavior that is dependent upon the specified Return Code ID: • ISDownloadProgress • ISPromptForSourceMedia • ISFilesInUse • ISRMFilesInUse • ISUpgradeParcel • ISSQLBrowse
Stop Event	This type of action allows you to conditionally stop further actions from being processed. For example, this action can be used to prevent the default behavior of a button from occurring.
Run a Suite Action	 ***Project*** • *This functionality is available in Suite/Advanced UI projects.* This type of action launches an action in the installation. The action can be any of the ones that are defined in the Events view of the Suite/Advanced UI project. For more information, see Using Actions to Extend the Behavior of a Suite/Advanced UI Installation.

Table 4-38 • Control Actions (cont.)

Type of Action	Description
Load Web Deploy Publish Profile	 ***Project*** • *This functionality is available in Suite/Advanced UI projects.* This type of action launches a secondary window that lets end users browse to and load a publisher profile file (.publishsettings) that contains the configuration settings that they want to use for the Web Deploy package. For more information, see Adding a Web Deploy Package to a Suite/Advanced UI Project.

Table 4-38 • Control Actions (cont.)

Type of Action	Description
Database	 ***Project*** • *This functionality is available in Suite/Advanced UI projects.* The following database-related options are available: • **Configure Database Metadata**—Initializes the database metadata (connection string, ODBC driver, timeout value, etc.) that will be used for connecting to a database, and saves the name of the configuration in an extension DLL for other actions to reference as the active configuration. • **Override SQL Login Properties**—Defines Suite properties that will be used to identify SQL login settings to establish a connection to a specified database server. For example, when the **Validate a Database Login** action is called, the Suite SQL runtime reads the value of the Suite properties specified by the **Override SQL Login Properties** action to retrieve the SQL login settings that are entered on a Suite UI wizard page and then attempts to establish a connection to the specified database server. • **Browse for Database Catalogs**—Retrieves the list of available database catalogs and saves the result to a Suite property (**ISSQLBrowseList**). This action uses metadata and properties identified by the **Configure Database Metadata** and **Override SQL Login Properties** actions. • **Browse for Database Servers**—Retrieves the list of available database servers and saves the result to a Suite property ((**ISSQLBrowseList**). This action uses metadata and properties identified by the **Configure Database Metadata** and **Override SQL Login Properties** actions. • **Browse for Registered Database Servers**—Retrieves a list of database servers registered in ODBC DSN and saves the list to a Suite property (**ISSQLBrowseLocalList**). This action uses metadata and properties identified by the **Configure Database Metadata** and **Override SQL Login Properties** actions. • **Validate a Database Login**—Adds a Validate action to verify the database login credentials using database metadata and properties identified by **Configure Database Metadata** and **Override SQL Login Properties**. • **Run a SQL String**—Executes a SQL statement using database metadata and properties identified by **Configure Database Metadata** and **Override SQL Login Properties**.

Table 4-38 • Control Actions (cont.)

Type of Action	Description
Browse for DLL Action	This type of action calls a function in a DLL file that you have created and added to your project through the Support Files view. For example, you can use a DLL file to trigger custom behavior.

Using Images, Text Files, and Other Objects in the Wizard Interface



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Any images, text files (such as End-User License Agreements), and other files that you want to use in the wizard interface of your Advanced UI or Suite/Advanced UI project are included in your project as support files. Support files are files that are installed to the target system for use only during the installation process. These files are automatically removed from target system when the installation is complete.

If your project supports multiple languages, you can add language-specific support files to the appropriate language-specific areas of the Support Files view. At run time, the Advanced UI or Suite/Advanced UI installation displays the appropriate language-specific support files.

For more information, see Using Support Files.

DPI-Awareness in the Wizard Interface



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

DPI-awareness in Advanced UI and Suite/Advanced UI projects has two parts:

- Items that are drawn by the engine are scaled according to the target system's DPI settings. Thus, if a target system has a DPI of 200%, the check box control is scaled accordingly.

- The engine considers the scale factor and language when displaying images and other resources that you are including in the wizard interface.

For example, when the engine determines that the scale factor should be 150 and that the appropriate language is English, it looks for resources that have the string *scale-150* in their path or file name. It searches the following paths in the order listed when searching for **image.png** on the target system:

1. [SUPPORTDIR]0409**scale-150**\image.png
2. [SUPPORTDIR]0409\image.**scale-150**.png
3. [SUPPORTDIR]0409\image.png
4. [SUPPORTDIR]**scale-150**\image.png
5. [SUPPORTDIR]image.**scale-150**.png
6. [SUPPORTDIR]\image.png

Note that matching the language takes precedence over matching the correct DPI.

InstallShield includes scale-150 and scale-200 images for the built-in default images that are included in Advanced UI and Suite/Advanced UI projects. If you want to include custom resources in your project, use the Support Files view to add appropriately scaled images.

Localizing the End-User Interface



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

InstallShield has many settings that let you specify text strings that should be displayed to end users. For example, when you enter a value for the Display Name and Description settings for a feature, you are configuring the feature name and description that may be displayed in the CustomSetup dialog at run time. Whenever you enter a value for such a setting that may need to be localized for one or more languages, InstallShield automatically creates a string entry for each language that your project supports. Each string entry consists of a language-independent identifier and a corresponding language-specific value. At run time, the installation displays the appropriate translated string values. Using string entries instead of hard-coded text enables you to keep the code for your installation completely separate from any language-specific strings that you may want to be displayed during the installation.

To help streamline the process of localizing a project, all of the text strings that may be displayed at run time during the installation process are available in one consolidated view: the String Editor view. You can use this view to edit the strings for everything from button text to feature descriptions. You can also use this view to export each language's string entries to a file, translate the values that are listed in the file, and then import the translated file into your project.

Working with String Entries in Projects that Support Multiple Languages



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*



Edition *• The Premier edition of InstallShield includes support for creating multilingual installations.*



Note *• To add additional languages to your project, use the Setup Language setting in the General Information view.*

Your project contains a string entry for every localizable text string and for each language that your project supports. The String Editor view shows the collection of language-independent identifiers and corresponding language-specific values for your project.

Note the following details about string entries in projects that include support for multiple languages:

- The values for all of the localizable settings throughout InstallShield—such as the Display Name and Description settings for a feature—use the project's default language (which you can specify in the String Editor view). If you edit the value of one of these localizable settings, InstallShield simultaneously updates the string entry's value in the String Editor view for the default language.

  Likewise, if you use the String Editor view to modify a string value for your project's default language, InstallShield simultaneously updates the settings in the other InstallShield views that display the value of that string entry.

- Each language that is supported by your project uses the same set of string identifiers.

  Therefore, if you add a string entry to a project, InstallShield uses the same string identifier for each language. If you rename a string identifier, InstallShield renames the string identifier for all languages. If you delete a string entry, InstallShield deletes it from each language.

- If you add a new language to your project, InstallShield adds to that language the same string identifiers that are available in the other languages that are in your project. For default string entries—that is, those that are available for the built-in dialogs and other user-interface elements—InstallShield adds the already-translated string values. For custom string entries that you have created, InstallShield uses the string values from the default language for the new language. You can modify any string values as needed for the new language.

When you finish authoring your project, you can export the string entries for each language to text files, and send the text files out for translation. Once the string entries in the text file are translated, you can import them into your project.

Using String Entries in InstallShield

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

Note • *Localization of UWP app packages requires the Windows 10 SDK be installed on the same machine as InstallShield. If it is not present, InstallShield will build a UWP app package (.appx) containing only the default language.*

Instead of hard-coding strings throughout your project, you can use string entries in areas of InstallShield that accept localizable text. The manner in which you select a string entry differs depending on where you need to use it.

Settings in Various Views

When you are entering the value of a setting in one of the views in InstallShield and that value is a text string that can be presented to end users, InstallShield automatically uses a string identifier for that setting. InstallShield places the string identifier in curly brackets before the string value. Following is an example of the value for the Display Name setting of a feature:

{ID_STRING24}My New Feature

In this example, **ID_STRING24** is the string identifier that is used for the Display Name setting. **My New Feature** is the string value for the project's default language.

Task ***To work with a setting that accepts a localizable text string entry, do one of the following:***

- To enter a new text string that is not present anywhere else in the project, type the text string that you want to use. InstallShield automatically assigns a new string identifier to the value that you enter.

  If your project includes support for multiple languages, InstallShield adds the new string identifier to all of your project's languages, and uses the string value that you entered as the value for all languages.

- To look at a list of existing string entries that are used in the project, click the ellipsis button (...) that is displayed on the right when you click the setting's value. InstallShield opens the Select String dialog box, which lets you select an existing string entry or create a new string entry.

Caution • *Do not confuse editing an existing string value with entering a new string identifier by entering text in an InstallShield setting. Once you have selected a string identifier for a setting, deleting the localizable text in the setting merely deletes the current string value; it does not replace the current string identifier with a new one.*

Dialog Editor

Many of the controls in the Dialog Editor accept strings that are displayed to the end user. For example, the Text and Tooltip properties always require localizable text.

Using a string entry in the Dialog Editor is similar to selecting string identifiers in other areas of InstallShield.



Task ***To work with the Dialog Editor and a control's property that uses localizable text, do one of the following:***

- Click the control's property sheet and edit the string. Doing so changes the value of the string entry for the current language. When you enter a text value without first selecting a string, InstallShield creates a new string identifier for your project.

  If your project includes support for multiple languages, InstallShield adds the new string identifier to all of your project's languages, and uses the string value that you entered as the value for all languages.

- When you click a localizable property's value to edit it, an ellipsis button (**...**) appears inside the property sheet. Click the ellipsis button to view the string entries for the current language.

The major difference between the dialog control properties and the settings in other views in InstallShield is that the other views always display the string value for the default language. When you edit a dialog's layout, however, you must first select the language of the dialog. Then, all strings displayed in the dialog and in the property sheet are from the current language of the dialog that you are editing.

InstallScript Script Editor Pane

If you are working in the InstallScript script editor pane in the InstallScript view, you can select and edit string entries through the Select String dialog box.



Task ***To use a string identifier in your InstallScript:***

1. Place the cursor at the point in your script where you want the string identifier to be inserted.
2. On the **Edit** menu, point to **Insert** and click **String Entry**. The **Select String** dialog box opens.
3. Do one of the following:
   - To use an existing string entry, click the row that contains the string entry that you want to use in your script.
   - To create a new string entry, click the **New** button. The **String Entry** dialog box opens, enabling you to create a new string identifier and value.
4. Click **OK**.

InstallShield places the string identifier in your script preceded by the at (@) symbol.

For more information, see String Constant Operator (@).



Tip *• Although you can include hard-coded user-interface strings directly in InstallScript code, it is recommended that you avoid doing this. Strings that are hard coded in InstallScript code are not stored as Unicode; thus, they are displayed correctly only when the installation is run on systems that have the correct code page. Adding strings to a project*

through the String Editor view and referencing the associated string identifiers from InstallScript code eliminates this issue.

What Happens with String Entries at Build Time and Run Time

InstallShield stores string entries in the **ISString** table of your InstallShield project file (.ism). At build time, InstallShield uses the string values instead of the string identifiers when it is creating your release in most cases. That is, the string identifiers are not built into the release, and the string identifiers are not available at run time.

The one exception of when InstallShield does use string identifiers in a build is if your project includes InstallScript code. In this scenario, the release that InstallShield builds includes string tables that contain all of the string entries. These string tables make it possible for you to use string identifiers in your InstallScript code instead of hard-coded text strings. At run time, the installation replaces string identifiers with string values as needed.

Adding a String Entry



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The String Editor view lets you create a new string entry that you can later reference in one of the settings in InstallShield, in one of your project's dialogs, and in your InstallScript code.



Task ***To add a new string entry to your project:***

1. In the View List under **User Interface**, click **String Editor**.
2. Do one of the following:
   - Click the **New String Entry** button.
   - Press the INSERT key.

   The **String Entry** dialog box opens.
3. In the **String Identifier** box, specify the language-independent string identifier that you want to use for your string entry, or leave the new default string identifier that InstallShield enters in this box for you.
4. In the **Value** box, specify the localizable text string that you want to use for the string entry.
5. In the **Comments** box, you can optionally specify an internal note about the string entry. The comments are not displayed at run time.
6. Click **OK**.

InstallShield adds a new row in the String Editor view for the new string entry.

Tip • *If your project includes support for multiple languages, InstallShield adds the string entry to all of your project's languages, and it uses the string value that you entered as the value for all languages. You can edit the string values for languages as needed.*

Note • *Since you must use a string identifier throughout InstallShield for settings that accept localizable text, InstallShield adds a new string entry to your project whenever you try typing a value into a setting without selecting an existing string identifier.*

Editing a String Entry

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The String Editor view contains a spreadsheetlike table that lets you modify string entries in your project.

Task ***To edit a string entry in the String Editor view:***

1. In the View List under **User Interface**, click **String Editor**.
2. Find the string entry that you want to modify.
3. Do one of the following:
   - To overwrite all of the text in a table cell, click the table cell and then type your new identifier, value, or comments.
   - To place the cursor at a particular place within a table cell, double-click that place. Then type your change.

Note that if you rename a string identifier, InstallShield renames the string identifier for all languages in your project.

When you edit a string entry, InstallShield updates the Modified column with the date and time that you made the change. You can sort the string entries in the String Editor view by modified date. This is helpful if you want to identify which strings have been modified since the strings were last translated.



Tip • *If you are editing a localizable setting from within one of the other views in InstallShield, you can click the ellipsis button (...) in that setting. Doing so opens the Select String dialog box, which lets you specify which string entry you want to use for the selected setting.*



Project • *In Basic MSI, InstallScript MSI, and Merge Module projects, some of the string values contain Windows Installer properties inside square brackets—for example,* ***Install [ProductName]****. At run time, the property and brackets are replaced by the property value.*

String values may also contain font information in curly brackets—for example, ***{&MSSansBold8}OK****. The font information indicates style details that should be used to display the strings at run time.*

Searching for Instances of String Identifiers



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

String identifiers are used throughout InstallShield projects. If you want to see where a particular string identifier is used in your project, you can perform a search in the String Editor view.



Task ***To find all the occurrences of a particular string identifier within your project:***

1. In the View List under **User Interface**, click **String Editor**.
2. Select the row that contains the string identifier for which you want to search.

   To select multiple consecutive rows, click the first row, and then press SHIFT while clicking the last row. To select multiple nonconsecutive rows, click the first row, and then press CTRL while clicking each additional row.
3. Click the **Search for Selected Strings in Project** button.

InstallShield displays the search results on the Results tab in the Output window. The Results tab indicates how many times the string identifier is used. In addition, it indicates the location in the Direct Editor view where the string identifier is used.

If the string identifier is not used in the project, the Results tab shows that zero occurrences were found.



Note • *InstallShield does not search the project's InstallScript files (.rul) for instances of the string identifier.*

Finding and Replacing String Entries

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The String Editor includes support for standard find-and-replace functionality. This functionality may be useful if you want to search for specific strings in your project and replace them with revised strings.

Finding Strings in the String Editor

Task ***To find a string in the String Editor:***

1. In the View List under **User Interface**, click **String Editor**.
2. Click the **Find String** button. The **Find** dialog box opens.
3. In the **Find what** box, type the string that you want to find.
4. Select any other options that you want.
5. Click **Find Next**.

InstallShield finds the first instance of the string that you specified.

Finding and Replacing Strings in the String Editor

Task ***To find and replace a string in the String Editor:***

1. In the View List under **User Interface**, click **String Editor**.
2. Click the **Find and Replace** button. The **Replace** dialog box opens.
3. In the **Find what** box, type the string that you want to replace.
4. In the **Replace with** box, type the new string.
5. Select any other options that you want.
6. Click **Find Next**, **Replace**, or **Replace All**.

InstallShield replaces strings as needed.

Removing a String Entry



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

If you no longer need a particular string entry in your project, you can delete it.

Note that if you delete a string entry, InstallShield deletes it from each language in your project.



Task ***To remove a string entry from your project:***

1. In the View List under **User Interface**, click **String Editor**.
2. Find the string entry that you want to delete.
3. Click the **Delete Selected Strings** button or press DELETE.

Removing String Identifiers from Localizable Settings



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

When you are entering the value of a setting in one of the views in InstallShield and that value is a text string that can be presented to end users, InstallShield automatically uses a string identifier for that setting. InstallShield places the string identifier in curly brackets before the string value.

If you try to delete the setting's value, you are actually deleting the string value for the current language. In that case, the string identifier with an empty value is used for the setting.

Therefore, if you want to remove a string identifier from a localizable setting, you must replace it with a different string identifier, or add a new string identifier.

For information on replacing a string identifier, see Editing a String Entry. To learn how to create a new string identifier, see Adding a String Entry.

Displaying Billboards



***Project** • Billboards are available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

You can add billboards to your projects to display information to end users during the installation process. The billboards can be used to communicate, advertise, educate, and entertain end users. For example, billboards can present overviews on new features of the product being installed or other products from your company. Each billboard is a file that you or your company's graphics department creates for complete control over the look and feel of the file transfer.



***Project** • Support for billboards differs, depending on which project type you are using. To learn more about billboards, see the appropriate section:*

- Displaying Billboards in Basic MSI Installations
- Displaying Billboards in InstallScript and InstallScript MSI Installations

Displaying Billboards in Basic MSI Installations



***Project** • This information applies to Basic MSI projects.*

You can add billboards to your projects to display information to end users during the installation process. The billboards can be used to communicate, advertise, educate, and entertain end users. For example, billboards can present overviews on new features of the product being installed or other products from your company. Each billboard is a file that you or your company's graphics department creates for complete control over the look and feel of the file transfer.

Billboard File Types for Basic MSI Projects



***Project** • This information applies to Basic MSI projects.*

InstallShield supports different types of files for billboards:

- Adobe Flash application file (.swf)
- Images (.bmp, .gif, .jpg, and .jpeg)

If a target system does not have the Adobe Flash Player, which is used to display Flash application files, the installation can detect that and display image billboards in place of the Flash billboard. Therefore, if you include a Flash billboard in your project, it is recommended that you also include one or more image billboards in your project.

Note • *You can add more than one image billboard to a project, but only one Adobe Flash application file billboard.*

Types of Billboards for Basic MSI Projects

Project • *This information applies to Basic MSI projects.*

InstallShield offers support for three different billboard styles. For example, with one style, the installation displays a full-screen background, with billboards in the foreground, and a small progress box in the lower-right corner of the screen. With another style, the installation displays a standard-size dialog that shows the billboards. The bottom of this dialog shows the progress bar.

Following are descriptions and sample screen shots of each type of billboard.

Fullscreen with Small Progress (Displayed in Lower Right)

For the **Fullscreen with Small progress (displayed in lower right)** type of billboard, when the installation displays the standard end-user dialogs, it also displays a full-screen background. During file transfer, the installation shows full-screen backgrounds, with billboards in the foreground, and a small progress box in the lower-right corner of the screen.

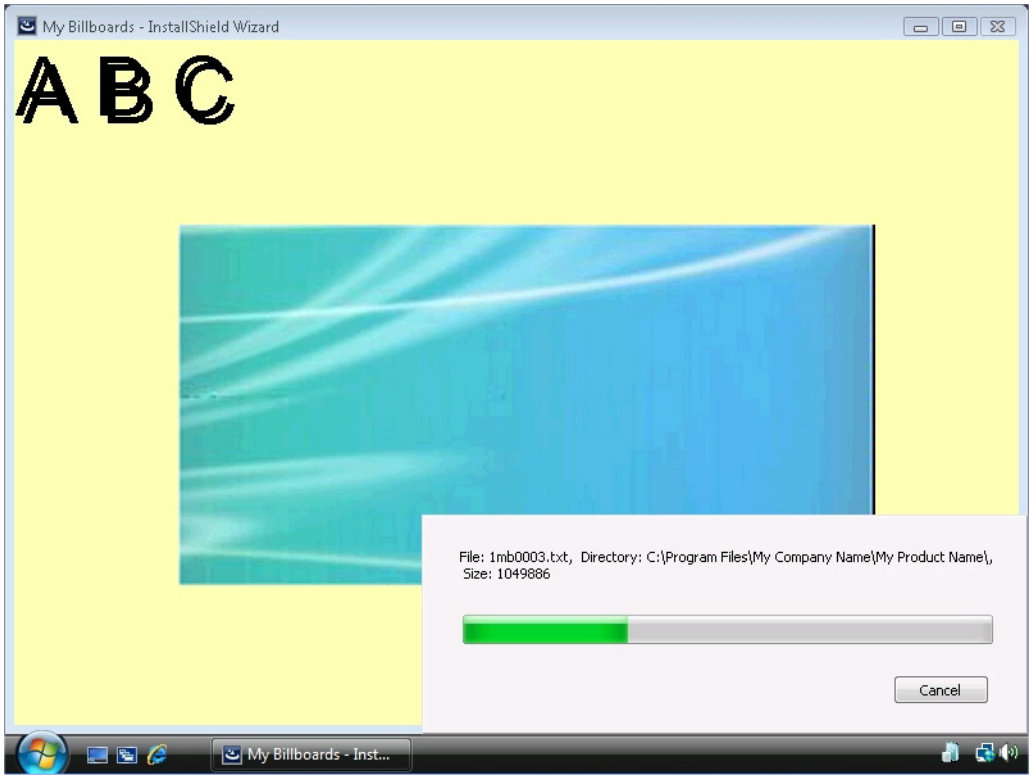


Figure 4-28: Fullscreen Billboard with Small Progress (Displayed in Lower Right)

In the sample screen shot, the billboard is the blue-green rectangle image in the center. Some of the configurable billboard settings were set as follows:

- Origin—Centered
- Title—A B C
- Font—48 pt. Arial
- Background Color—Yellow

Windowed with Standard Progress

For the **Windowed with Standard progress** type of billboard, during file transfer, the installation displays a standard-size dialog that shows the billboards. The bottom of this dialog shows the progress bar. The installation does not display a background for this style.

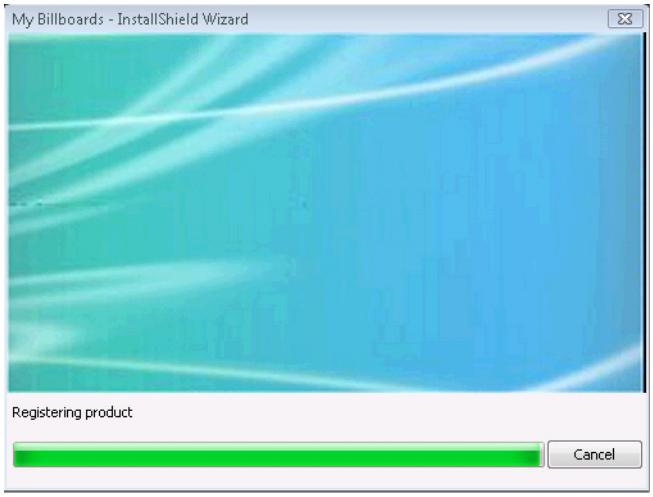


Figure 4-29: Windowed with Standard Progress

In the sample screen shot, the billboard is the blue-green rectangle image. Its size is 544 pixels wide by 281 pixels high.

Windowed with Small (Displayed in Lower Right, No Billboards)

For the **Windowed with Small (displayed in lower right, no billboards)** type of billboard, the installation displays a small progress box in the lower-right corner of the screen during file transfer. It does not display any billboards or a background.

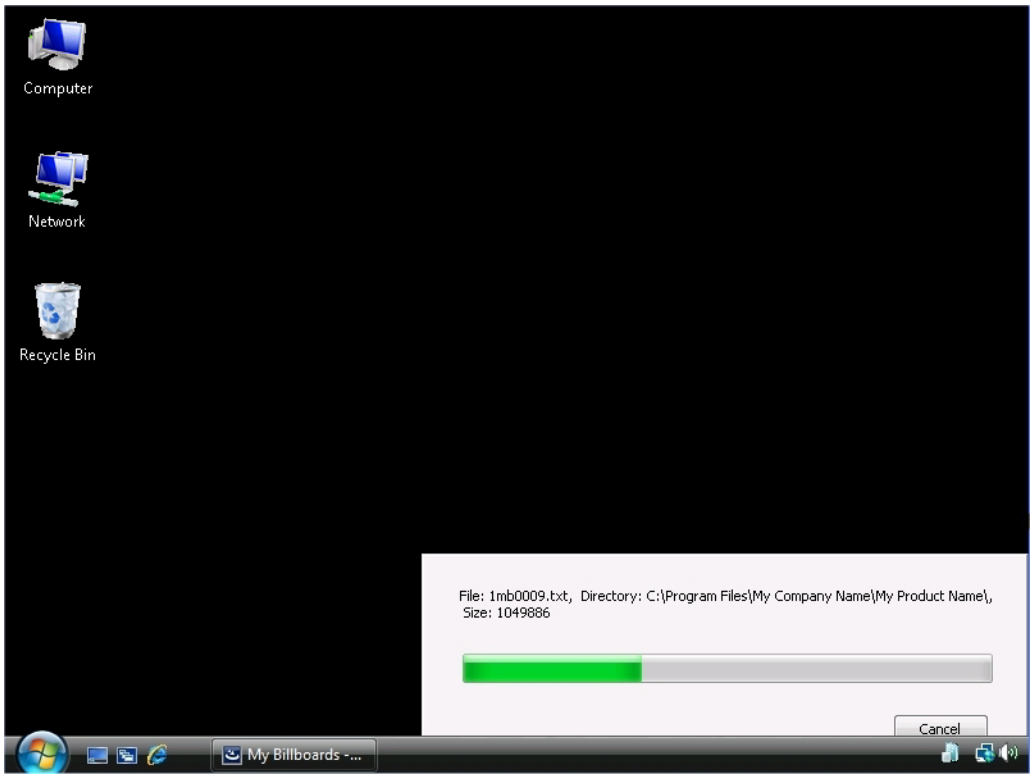


Figure 4-30: Windowed with Small (Displayed in Lower Right, No Billboards)

As shown in the sample screen shot, the progress bar is shown, but no billboard is displayed. The black background is the end user's desktop.

Specifying Which Type of Billboard to Use in a Basic MSI Project

Project • *This information applies to Basic MSI projects.*

InstallShield offers support for different billboard styles.

Task ***To specify which type of billboard you want to use in your installation:***

1. In the View List under **User Interface**, click **Billboards**.
2. In the center pane, click the **Billboards** explorer. InstallShield displays the **Billboard Type** setting in the right pane.
3. In the **Billboard Type** setting, select the appropriate style of billboard.

To see samples of each type of billboard, see Types of Billboards for Basic MSI Projects.

Adding an Adobe Flash Application File Billboard to a Basic MSI Project

Project • *This information applies to Basic MSI projects.*

InstallShield lets you display a Flash application file billboard during the file transfer process. Flash application files can consist of videos, movies, sounds, interactive interfaces, games, text, and more—anything that is supported by the .swf type of file. It is recommended that files such as Flash video files (.flv) and MP3 audio files be embedded in the .swf file so that they are available locally on the target system during file transfer. Although .swf files can reference external files that you can post on a Web site, this external implementation would require that end users have an Internet connection.

Task ***To add an Adobe Flash Application File billboard to your installation project:***

1. In the View List under **User Interface**, click **Billboards**.
2. In the **Billboards** explorer, right-click **Adobe Flash Application File (.swf)** and then click **New Billboard**. InstallShield adds a new billboard item with the name **NewBillboard1**.
3. Type a name for the billboard item. This name is used to help you identify the item while you are creating your installation; the name is not displayed during the installation.
4. In the right pane, configure the settings for the billboard.

Note • *If the version of Flash or other tool that you use to create your .swf file is newer than the version of the Flash Player that is installed on a target system, it is possible that some of the Flash features may not work as expected on that target system.*

If a target system does not have the Adobe Flash Player, which is used to display Flash application files, the installation can detect that and display image billboards in place of the Flash billboard. Therefore, if you include a Flash billboard in your project, it is recommended that you also include one or more image billboards in your project.

Adding Image Billboards to a Basic MSI Project



Project *• This information applies to Basic MSI projects.*

You can choose to display only one image billboard during the file transfer process, or you can include a series of image billboards, each one designed to be displayed for a specific amount of time. InstallShield includes support for .bmp, .gif, .jpg, and .jpeg image files.



Note *• Animated .gif files are not supported. If you want to use animation in a billboard, consider using an Adobe Flash application file billboard.*



Task ***To add an image billboard to your installation:***

1. In the View List under **User Interface**, click **Billboards**.
2. In the **Billboards** explorer, right-click **Images** and then click **New Billboard**. InstallShield adds a new billboard item with the name **NewBillboard1**.
3. Type a name for the billboard item. This name is used to help you identify the item while you are creating your installation; the name is not displayed during the installation.
4. In the right pane, configure the settings for the billboard.

Configuring Billboard Settings in a Basic MSI Project



Project *• This information applies to Basic MSI projects.*

When you add an Adobe Flash application file billboard or an image billboard to your project, you need to configure its settings.



Task ***To configure a billboard's settings:***

1. In the View List under **User Interface**, click **Billboards**.
2. In the **Billboards** explorer in the center pane, click the billboard that you want to configure. InstallShield displays the billboard settings in the right pane.
3. Configure the settings as required.

For information about each of the billboard settings, see Settings for Adobe Flash Application File Billboards and Image Billboards.

Previewing Billboards Without Building and Launching a Release

Project • *This information applies to Basic MSI projects.*

InstallShield lets you preview a billboard to see how it would be displayed at run time, without requiring you to build and run a release.

Previewing a billboard lets you see how your billboard will look with the background color, position, and related settings that are currently configured for your billboard.

Task ***To preview a billboard:***

1. In the View List under **User Interface**, click **Billboards**.
2. In the **Billboards** explorer in the center pane, right-click the billboard that you want to preview, and then click **Preview Billboard**.

InstallShield displays a preview of the billboard as it would be displayed at run time.

To stop previewing a billboard, click the Cancel button in the preview window.

Tip • *Previewing a billboard is especially helpful if you want to see how your Flash or image billboard will look with different selected billboard types. You can preview a billboard, change the billboard type, and then preview a billboard again.*

Setting the Billboard Order in a Basic MSI Project

Project • *This information applies to Basic MSI projects.*

Image billboards are displayed in the order in which they are listed in the Billboards view, from top to bottom.

Task ***To change the order in which image billboards are displayed at run time:***

1. In the View List under **User Interface**, click **Billboards**.
2. In the **Billboards** explorer, right-click one of the billboard items that you want to move, and then click either **Move Up** or **Move Down**.

Repeat the last step until all of the billboards are correctly sorted.

Run-Time Behavior of a Basic MSI Installation that Includes Billboards

Project • *This information applies to Basic MSI projects.*

Important *• If your installation includes billboards, your installation must include a* **Setup.exe** *setup launcher. The setup launcher is required because it displays the billboards at run time. The Setup.exe tab for a release in the Releases view is where you specify information such as whether you want to use a setup launcher. To learn more, see Setup.exe Tab for a Release.*

If your installation includes a Flash billboard and one or more image billboards, only one billboard type is displayed at run time during the file transfer process: the Flash billboard or the image billboards.

- If the Flash Player is present on the target system, the installation displays the Flash billboard.
- If the Flash Player is not present, the installation displays the image billboards.

The run-time behavior is slightly different, depending on whether the installation is displaying a Flash billboard or image billboards:

- **If the installation is displaying a Flash billboard**—When the file transfer is complete, the installation continues showing the Flash billboard until its duration has elapsed. Once the duration has elapsed, the installation stops displaying the billboard and shows the appropriate SetupComplete dialog.

  If the file transfer takes more time than you have allocated for the duration of the Flash billboard, the installation continues displaying the Flash billboard until file transfer ends.

- **If the installation is displaying image billboards**—When the file transfer is complete, the installation stops displaying the image billboards, even if other billboards are scheduled or the current billboard's duration has not elapsed. The installation then shows the appropriate SetupComplete dialog.

  If the file transfer takes more time than you have allocated for the billboards, the installation continues displaying the billboards until file transfer ends. If No is selected for the Loop Billboard setting in the Billboards view and the installation reaches the last billboard before the file transfer ends, the installation continues displaying that last image billboard until file transfer ends. Then the installation shows the appropriate SetupComplete dialog. If Yes is selected for this setting and the installation reaches the last billboard before the file transfer ends, the installation restarts the display of billboards from the beginning. The loop continues, if necessary, until the file transfer ends, and the SetupComplete dialog is displayed.

Note *• If the version of Flash or other tool that you use to create your .swf file is newer than the version of the Flash Player that is installed on a target system, it is possible that some of the Flash features may not work as expected on that target system.*

Removing a Billboard from a Basic MSI Project

Project *• This information applies to Basic MSI projects.*

Task ***To remove a billboard from your installation:***

1. In the View List under **User Interface**, click **Billboards**.
2. In the **Billboards** explorer, right-click the billboard that you would like to remove, and then click **Delete**.

Displaying Billboards in InstallScript and InstallScript MSI Installations

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

You can add billboards to your projects to display information to end users during the installation process. The billboards can be used to communicate, advertise, educate, and entertain end users. For example, billboards can present overviews on new features of the product being installed or other products from your company. Each billboard is a file that you or your company's graphics department creates for complete control over the look and feel of the file transfer.

Billboard Styles and File Types for InstallScript and InstallScript MSI Projects

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

InstallShield offers support for two different billboard styles. One includes a full-screen background window, and the other does not. Both styles show a progress bar.

Following are descriptions and sample screen shots of both types of billboard, along with applicable supported file types.

Windowed Billboards with Progress

For this style of billboard, the installation displays a standard-size dialog. During file transfer, the top of the dialog shows the billboards, and the bottom of this dialog shows the progress bar. The installation does not require a background window for this style of billboard.

You can use the following types of files for this style of billboard:

- Adobe Flash application file (.swf)
- Images (.bmp, .gif, .jpg, and .jpeg)

If a target system does not have the Adobe Flash Player, which is used to display Flash application files, the installation can detect that and display image billboards in place of the Flash billboard. Therefore, if you include a Flash billboard in your project, it is recommended that you also include one or more image billboards in your project.

Note *• For windowed billboards that show progress, you can add more than one image billboard to a project, but only one Adobe Flash application file billboard.*

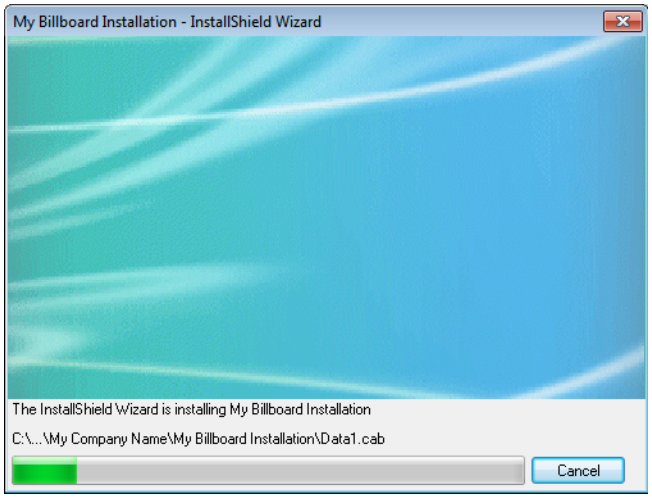


Figure 4-31: Windowed Billboard with Progress

In the sample screen shot, the billboard is the blue-green rectangle image. Its size is 544 pixels wide by 281 pixels high.

Note • *Skinned dialogs cannot display windowed billboards. To learn more about skins, see Dialog Skins.*

Billboards with a Full-Screen Background Window

For this style of billboard, when the installation displays the standard end-user dialogs, it also displays a full-screen background window. During file transfer, the billboards are displayed in front of the full-screen background window, and the standard size progress dialog is displayed in front of the billboards. This style of billboard requires a background window.

You can use image files (.bmp, .gif, .jpg, and .jpeg) for this style of billboard.

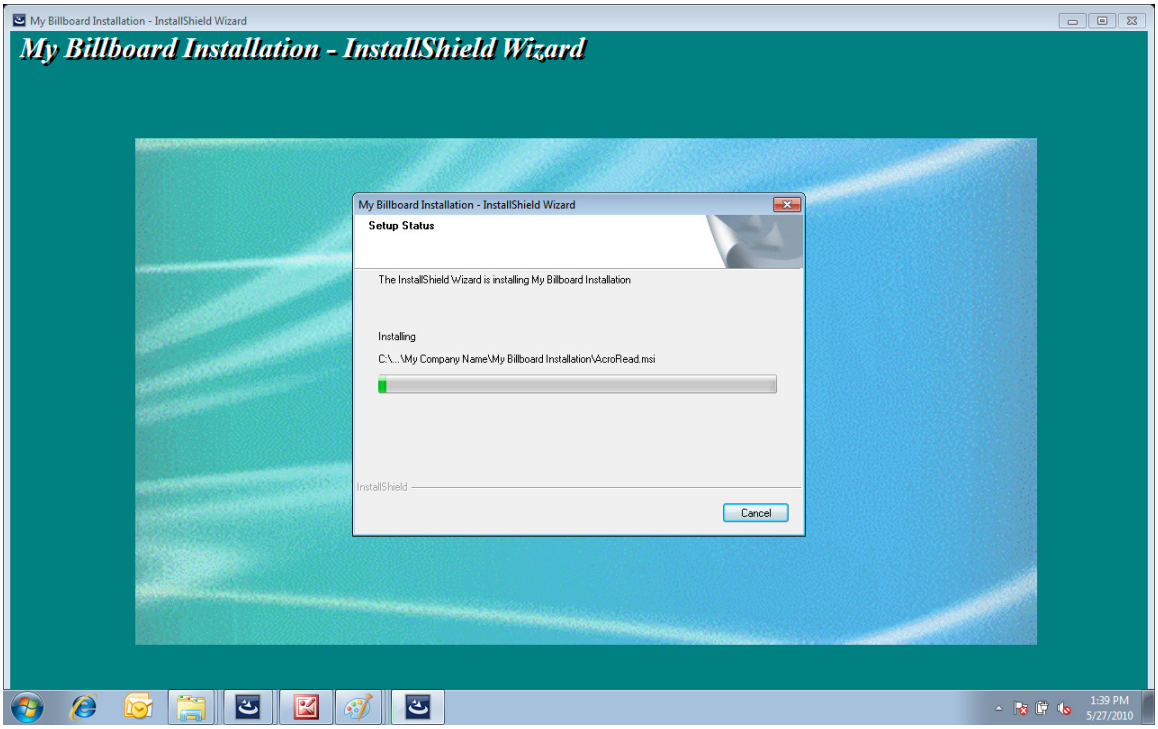


Figure 4-32: Billboard with Full-Screen Background Window

In the sample screen shot, the billboard is the blue-green rectangle image behind the progress dialog but in front of the background window.

Naming Billboard Files in an InstallScript or InstallScript MSI Project

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Note *• The Adobe Flash application file support that is described here is applicable if the billboard style that you are using is the windowed style with progress, not the billboard style with a full-screen background window. If you want to use Flash file support with a full-screen background window, use the* ***PlayMMedia*** *function; with this support, the following naming conventions do not apply to the Adobe Flash file. For more information, see PlayMMedia. To learn how to display the background window for this implementation, see Displaying a Background Window in InstallScript and InstallScript MSI Installations.*

You can add more than one image billboard to a project, but only one Adobe Flash application file billboard.

Note that if a target system does not have the Adobe Flash Player, which is used to display Flash application files, the installation can detect that and display image billboards in place of the Flash billboard. Therefore, if you include a Flash billboard in your project, it is recommended that you also include one or more image billboards in your project.

The first step in adding billboards to your project is to create the Adobe Flash application file (if applicable) and image files (.bmp, .gif, .jpg, or .jpeg) that serve as your installation's billboards. When your files are properly named, you can add your billboards to your project. Note that if a target system does not have the Adobe Flash Player, which is used to display Flash application files, the installation can detect that and display image billboards in place of the Flash billboard. Therefore, if you include a Flash billboard in your project, it is recommended that you also include one or more image billboards in your project.

Billboard files must follow a designated naming convention. Each file name must begin with **bbrd**, followed by the number of the billboard (from 1 through 99); each must end with a supported file extension (.swf, .bmp, .gif, .jpg, or .jpeg). You can include a maximum of one Adobe Flash file (.swf) in your project; if you do include an .swf file, its name must contain the lowest number in the list of billboard files (for example, **bbrd1.swf**) that you are including in your project.

The billboards are displayed according to the sequential order of their file names. For example, a billboard file named **bbrd2.bmp** is displayed before **bbrd4.gif**.

Note that it is not necessary for the file name numbers to be contiguous; that is, you can add **bbrd1.jpg**, **bbrd3.jpg**, and **bbrd5.jpg** to your project, and each image is displayed at run time in order.

Tip *• The length of time that an image billboard is displayed depends upon the number of image billboards in your installation project. The percentage of the display time is approximately 1 divided by the number of billboards. For example, if you have four billboards in your installation, each billboard is displayed for 25% of the file-transfer time.*

Adding an Adobe Flash Application File Billboard to an InstallScript or InstallScript MSI Project

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Note *• The Adobe Flash application file support that is described here is applicable if the billboard style that you are using is the windowed style with progress, not the billboard style with a full-screen background window. If you want to use Flash file support with a full-screen background window, use the* ***PlayMMedia*** *function. For more information, see PlayMMedia. To learn how to display the background window for this implementation, see Displaying a Background Window in InstallScript and InstallScript MSI Installations.*

InstallShield lets you display a Flash application file billboard during the file transfer process. Flash application files can consist of videos, movies, sounds, interactive interfaces, games, text, and more—anything that is supported by the .swf type of file. It is recommended that files such as Flash video files (.flv) and MP3 audio files be embedded in the .swf file so that they are available locally on the target system during file transfer. Although .swf files can reference external files that you can post on a Web site, this external implementation would require that end users have an Internet connection.

Task ***To add an Adobe Flash Application File billboard to your installation project:***

1. In the View List under **Behavior and Logic**, click **Support Files/Billboards**.
2. In the **Support Files** explorer, click the **Billboards** item that should contain the billboard: either **Language Independent** or a language-specific item. The **Files** pane is displayed on the right.
3. Right-click anywhere in the **Files** pane and click **Insert Files**, or place your cursor in the **Files** pane and press the Insert key. The **Open** dialog opens.
4. Select your Adobe Flash application file billboard file named **bbrd1.swf**, and click **OK**.

InstallShield adds the file to your project.

Note *• If the version of Flash or other tool that you use to create your .swf file is newer than the version of the Flash Player that is installed on a target system, it is possible that some of the Flash features may not work as expected on that target system.*

If a target system does not have the Adobe Flash Player, which is used to display Flash application files, the installation can detect that and display image billboards in place of the Flash billboard. Therefore, if you include a Flash billboard in your project, it is recommended that you also include one or more image billboards in your project.

Adding an Image Billboard to an InstallScript or InstallScript MSI Project

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

You can choose to display only one image billboard during the file transfer process, or you can include a series of image billboards, each one designed to be displayed for a specific amount of time. InstallShield includes support for .bmp, .gif, .jpg, and .jpeg image files.

Note *• Animated .gif files are not supported. If you want to use animation in a billboard, consider using an Adobe Flash application file billboard.*

Task ***To add an image billboard to your installation project:***

1. In the View List under **Behavior and Logic**, click **Support Files/Billboards**.
2. In the **Support Files** explorer, click the **Billboards** item that should contain the billboard: either **Language Independent** or a language-specific item. The **Files** pane is displayed on the right.
3. Right-click anywhere in the **Files** pane and click **Insert Files**, or place your cursor in the **Files** pane and press the Insert key. The **Open** dialog opens.
4. Select a billboard file, and click **OK**.

InstallShield adds the file to your project.

Tip *• The length of time that an image billboard is displayed depends upon the number of image billboards in your installation project. The percentage of the display time is approximately 1 divided by the number of billboards. For example, if you have four billboards in your installation, each billboard is displayed for 25% of the file-transfer time.*

Adding or Modifying the Code in an InstallScript or InstallScript MSI Project to Display Billboards

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The InstallScript code that is needed to display billboards at run time varies, depending on the style of billboard that you are using.

- **Windowed Billboards with Progress**—To use windowed billboards that show progress at run time, use the STATUSBBRD constant with the **Enable** function.
- **Billboards with a Full-Screen Background Window**—To use billboards that are displayed with a full-screen background window, use the FULLWINDOWMODE and BACKGROUND constants with the **Enable** function. To learn how, see Displaying a Background Window in InstallScript and InstallScript MSI Installations.

  Note that if you want to use an Adobe Flash billboard with this style of billboard, you must also use the **PlayMMedia** function.

For details about these different styles of billboards, see Billboard Styles and File Types for InstallScript and InstallScript MSI Projects.

Setting the Billboard Order in an InstallScript or InstallScript MSI Project

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Image billboards are displayed according to the numeric order of their file names. A billboard file named **bbrd2.bmp** is displayed to the end user before a file named **bbrd3.bmp**, and after a file named **bbrd1.bmp**. For specific guidelines on naming conventions, see Naming Billboard Files in an InstallScript or InstallScript MSI Project.

Task ***To change the order in which billboards are displayed:***

1. Remove the billboard files whose order should change. (For more information, see Removing a Billboard from an InstallScript or InstallScript MSI Project.)
2. Using Windows Explorer, rename the billboard files so that they are in the appropriate numeric order.
3. Add the renamed files, as explained in Adding an Image Billboard to an InstallScript or InstallScript MSI Project.

Note that it is not necessary for the file name numbers to be contiguous; that is, you can add **bbrd1.jpg**, **bbrd3.jpg**, and **bbrd5.jpg** to your project, and each image is displayed at run time in order.

Displaying Billboards with Special Effects During an InstallScript or InstallScript MSI Installation



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The **SetDisplayEffect** function enables you to display your billboards with different special effects when they first appear on the main installation window. Choose one of this function's options before you display a bitmap with **PlaceBitmap** or display billboards during file transfer.

Moving Billboards to a Different Screen Location for an InstallScript or InstallScript MSI Project



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*



Note *• The Adobe Flash application file support that is described here is applicable if the billboard style that you are using is the billboard style with a full-screen background window (which uses the* ***PlayMMedia*** *function), not for the windowed style with progress (which uses the STATUSBBRD constant with the* ***Enable*** *function). For more information about these two styles of billboards, see Billboard Styles and File Types for InstallScript and InstallScript MSI Projects.*

By default, the installation displays billboards in the center of the main installation window. To specify a different location, call the **PlaceWindow** function with the BILLBOARD option. For example, to place your billboards 10 pixels from the upper-left corner of the screen, make the following call:

```
PlaceWindow (BILLBOARD, 10, 10, UPPER_LEFT);
```

Since billboards are displayed only during file transfer, ensure that you call **PlaceWindow** before you begin file transfer.

Removing a Billboard from an InstallScript or InstallScript MSI Project



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Task ***To remove a billboard from your installation project:***

1. In the View List under **Behavior and Logic**, click **Support Files/Billboards**.
2. In the **Support Files** explorer, click the **Billboards** item that contains the billboard that you want to remove: either **Language Independent** or a language-specific item. The **Files** pane is displayed on the right.
3. Right-click the billboard file and click **Delete**.

Displaying a Background Window in InstallScript and InstallScript MSI Installations

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

By default, installations do not display a background window. If you want to use the **PlayMMedia** function to display an Adobe Flash application file (.swf) or AVI files, your installation must display a background window. You may also want to display a background window for one type of billboard.

Tip • *You can display a Flash file billboard and image billboards without a background window. For more information, see Billboard Styles and File Types for InstallScript and InstallScript MSI Projects.*

The method for displaying a background window depends on which project type you are using.

InstallScript Projects

To display a background window in an InstallScript project, modify the script's **OnShowUI** event handler by removing the double slashes from the beginnings of the following lines of code:

```
//if ( LoadStringFromStringTable( "TITLE_MAIN", szTitle ) < ISERR_SUCCESS ) then // Load the title
string.
//    szTitle = IFX_SETUP_TITLE;
//endif;
//SetTitle( szTitle, 24, WHITE );
//Enable( FULLWINDOWMODE );
//Enable( BACKGROUND );
//SetColor( BACKGROUND, RGB( 0, 128, 128 ) );
```

InstallScript MSI Projects

To display a background window in an InstallScript MSI project, modify the appropriate script's UI event handler (such as **OnFirstUIBefore** or **OnMaintUIBefore**) by removing the double slashes from the beginnings of the following lines of code:

```
// SetTitle( @PRODUCT_NAME, 24, WHITE );
// SetTitle( @PRODUCT_NAME, 0, BACKGROUNDCAPTION );
// Enable( FULLWINDOWMODE );
```

```
// Enable( BACKGROUND );
// SetColor(BACKGROUND,RGB (0, 128, 128));
```

Populating List Boxes at Run Time

Basic MSI Projects

Populating a list box at run time requires you to create temporary records for the .msi database using SQL queries. For an example, see Knowledge Base article Q103295. For additional information, see InstallShield Developer Tip: Accessing the MSI Database at Run Time.

InstallScript MSI Projects

The **CtrlSetList** function associates a string list variable with a ListBox control on a dialog. For example:

```
function FillListBox( )
    LIST listDays;
begin
    listDays = ListCreate(STRINGLIST);
    ListAddString(listDays, "Monday", AFTER);
    ListAddString(listDays, "Wednesday", AFTER);
    ListAddString(listDays, "Friday", AFTER);

    CtrlSetList("DialogName", nListBoxId, listDays);
end;
```

Displaying File Browse Dialogs

For an example of calling the **GetOpenFileName** API to display a file browse dialog, see Knowledge Base article Q104325.

Displaying Network Browse Dialogs in InstallScript Installations

You can use the **SelectDirEx** function to display a dialog with which the user can browse the Network Neighborhood.

```
prototype NetBrowse( );

function NetBrowse( )
    STRING svSelectedDir[MAX_PATH + 1];
    NUMBER nReturn;
begin
    svSelectedDir = PROGRAMFILES;

    nReturn = SelectDirEx("", "Select a directory:", "", "",
        BIF_EDITBOX | BIF_RETURNONLYFSDIRS, svSelectedDir);

    if (nReturn = OK) then
        MessageBox("Selected directory was: " + svSelectedDir, INFORMATION);
    elseif (nReturn = CANCEL) then
        MessageBox("User clicked Cancel", INFORMATION);
```

```
    else
        MessageBox("Error displaying dialog", WARNING);
    endif;
end;
```

Preparing Installations for Update Notifications

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

FlexNet Connect provides a mechanism that enables you to automatically notify your Web-connected end users when patches, updates, and product information for your application are ready for release. FlexNet Connect helps you reduce the number of end users running old releases of your products and prevent them from installing the wrong updates from your Web site.

FlexNet Connect Implementation

There are two cycles of tasks that you perform when using FlexNet Connect to automatically inform end users about updates: initial deployment and update deployment. Once you have performed the initial deployment steps for your application, each time you want to distribute an update of that application to your customers, you perform the update deployment cycle of steps. To learn about the update deployment steps, see Notifying End Users about Upgrades Using FlexNet Connect.

Initial Deployment

1. Use InstallShield to create the installation project for your application. FlexNet Connect should be enabled in this project. When you enable FlexNet Connect, InstallShield includes the Software Manager in your installation. This desktop tool ships with your application and provides a tool for end users to view available updates.
2. Register your application with the FlexNet Connect Publisher site, a Web-based management portal.
3. Install your application and test it.

FlexNet Connect includes a variety of options that can be purchased together for a complete solution, or separately for a customized solution. To learn more, visit the Flexera Software Web site.

Enabling Automatic Update Notifications for a Project

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

Caution • *Enabling automatic update notifications in your project adds about 600 KB of files to your installation. These files must be distributed with your application in order for FlexNet Connect to work. If you cannot afford to include these files in your installation because of bandwidth limitations or other reasons, you can select* ***No*** *to disable automatic update notifications. However, FlexNet Connect cannot be used to deploy an update unless automatic*

*notification is enabled in the original installation when you distribute it to your end users. Therefore, if you select **No**, you will not be able to later take advantage of the automatic update notification features.*



Task ***To enable automatic update notification for your project:***

1. In the View List under **Installation Information**, click **Update Notifications**.
2. For the **Enable FlexNet Connect** setting, select one of the **Yes** options.
3. To register your product with FlexNet Connect, select the **Is Product/Version Registered** setting and follow the directions in the help pane.

Disabling Automatic Update Notifications for a Project



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.



***Caution** • Enabling automatic update notifications in your project adds about 600 KB of files to your installation. These files must be distributed with your application in order for FlexNet Connect to work. If you cannot afford to include these files in your installation because of bandwidth limitations or other reasons, you can select **No** to disable automatic update notifications. However, FlexNet Connect cannot be used to deploy an update unless automatic notification is enabled in the original installation when you distribute it to your end users. Therefore, if you select **No**, you will not be able to later take advantage of the automatic update notification features.*



Task ***To disable automatic update notification for your project:***

1. In the View List under **Installation Information**, click **Update Notifications**.
2. For the **Enable FlexNet Connect** setting, select **No**.

InstallShield removes the FlexNet Connect files from your project.

Removing the Shortcut

If you added to your installation a shortcut that calls FlexNet Connect, you need to manually remove it.



Task ***To remove the shortcut:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. Right-click the shortcut and select **Delete**, or select the shortcut and press the **Delete** key.

Removing the Check-for-Updates Check Box from the Last Dialog

The following instructions describe how to remove the **Yes, check for program updates (Recommended) after the setup completes** check box from the **SetupComplete** dialog.

Task ***To remove the check-for-updates check box from the SetupComplete dialog:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Right-click the **ISENABLEDWUSFINISHDIALOG** property and select **Delete Property**.

Files that Need to Be Installed for Automatic Update Notification

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

To install with your application the ability to implement automatic notification of updates to your application, enable FlexNet Connect for your project. This is disabled by default for all new installation projects that you create.

When you enable FlexNet Connect in a Basic MSI or InstallScript MSI project, the FlexNet Connect merge module is added to your installation. When you enable FlexNet Connect in an InstallScript project, this merge module is added to your installation at build time.

The FlexNet Connect merge module includes several files that are installed on the target machine during installation of your application. These files must be distributed with your application in order for FlexNet Connect to work. Among the items that are installed:

- **Software Manager** (**ISUSPM.exe**) is an application that your end users can use to check for updates and product information. When an update is available for your application, the update is listed in the Software Manager, along with hyperlinks to download the update and view release notes.
- **Update Agent** (**Agent.exe**) is the component that handles all of the communication between the Software Manager and the notification server. Optionally, you can embed calls directly from your application to the Agent to create an update experience that is more integrated with your application.

Creating a Shortcut to Check for Updates

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

You can create a shortcut to launch FlexNet Connect.

Task ***To create a shortcut to check for updates:***

1. In the View List under **System Configuration**, click **Shortcuts**.
2. In the **Shortcuts** explorer, right-click one of the destination directories and select **New Shortcut to Preexisting file**.
3. Configure the following settings for the shortcut:

Table 4-1 • Settings for a Shortcut that Checks for Updates

Setting	Value
Display Name	Check for Updates
Target	[ALLUSERSPROFILE]FlexNet\Connect\11\agent.exe
Icon File	<ISProductFolder>\redist\Language Independent\OS Independent\UpdateService.ico
Arguments	/sn[ProductCode]
Key Name	Check for Updates

Adding a Check-for-Updates Check Box to the SetupComplete Dialog

Project • *This information applies to Basic MSI projects.*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

You can add a check-for-updates check box to the last dialog of your installation. If the end user selects this **Yes, check for program updates (Recommended) after the setup completes** check box and then clicks the **Finish** button to end the installation, FlexNet Connect is launched.

Task ***To add a check-for-updates check box to the SetupComplete dialog:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Set the **ISENABLEDWUSFINISHDIALOG** property to **1**.

Registering Your Application with FlexNet Connect

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

FlexNet Connect uses the product code and product version to uniquely identify products. Before you can properly test FlexNet Connect and automatic update notifications, you need to register the product code and product version of your application with FlexNet Connect. If you run FlexNet Connect before registering, your end users receive a "product not registered" message.

Task ***To register the product code and product version:***

1. In the View List under **Installation Information**, click **Update Notifications**.
2. Set the **Enable FlexNet Connect** property to **Yes** if you have not already done so.
3. Click the **Is Product/Version Registered** property. The help window displays instructions for setting this property. Follow the instructions there to complete the registration process.

Configuring Servers

When you are creating an installation, you may find it necessary to provide server-side support for some technology that will be installed on a target system. InstallShield makes it easy to configure server-side installations or manage COM+ server applications and application proxies. InstallShield provides support for Internet Information Services, SQL, and component services.

Configuring SQL Support



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*



Note *• For information about configuring SQL support in Suite/Advanced UI projects, refer to Adding a SQLLogin Predefined Wizard Page in a Suite/Advanced UI Project.*

InstallShield provides SQL support for Microsoft SQL Server, Microsoft Windows Azure, MySQL, and Oracle. The SQL Scripts view provides a central location for managing and organizing all SQL scripts by server connections and settings within the user interface. SQL support within InstallShield enables you to do the following:

- Connect to SQL servers.
- Import catalog schema and/or data.
- Associate SQL scripts with features.
- Set required SQL server/script properties (server name, database name, authentication method, etc.).
- Set SQL script for execution during installation or uninstallation.
- Edit SQL scripts.
- Require and/or target specific versions of SQL Server, MySQL, or Oracle.
- Define SQL script text replacement.
- Open scripts in Microsoft SQL Server Management Studio or Microsoft SQL Server Query Analyzer.



Note *• The import database functionality applies to the Microsoft SQL Server Database. Oracle users should refer to the Oracle Web page on Oracle Database Utilities for information on utilities that may work in conjunction with InstallShield.*

If you have Microsoft SQL Server Management Studio or Microsoft SQL Server 2000 SQL Query Analyzer installed on your system, you can open a new SQL script that you have added to your project to test, edit, and syntax-check the script. To launch one of those tools and open your script from within InstallShield, right-click the script file in the SQL

Scripts view and then click Open Script in Microsoft SQL Server Management Studio. InstallShield searches your system for the following tools in order and launches the first one that it finds:

1. *Microsoft SQL Server 2008 Management Studio (any edition, including Express;* **ssms.exe***)*
2. *Microsoft SQL Server 2005 Management Studio (***SqlWb.exe***)*
3. *Microsoft SQL Server 2005 Management Studio Express (***ssmsee.exe***)*
4. *Microsoft SQL Server 2000 Query Analyzer (***isqlw.exe***)*

Using Windows Installer Properties for SQL Login Settings



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

When you use the SQL Scripts view to add a new SQL database connection to your project, InstallShield adds the following Windows Installer properties to the project by default.

Table 4-1 • Default SQL Login Properties

Property	Description
IS_SQLSERVER_AUTHENTICATION	This property identifies the type of authentication that you want to use to connect to the specified catalog. The default property is **IS_SQLSERVER_AUTHENTICATION**. The following numbers are valid property values: • **0**—Windows authentication credential of the current user • **1**—Server authentication This property is the default value for the Authentic Type Property Name setting on the Advanced tab for the selected SQL connection.
IS_SQLSERVER_DATABASE	This property identifies the name of the SQL catalog to which you want to create a connection during the installation. This property is the default value for the Target Catalog Property Name setting on the Advanced tab for the selected SQL connection.
IS_SQLSERVER_PASSWORD	This property identifies the password that should be used for server authentication. This property is the default value for the Server Authentication Password Property Name setting on the Advanced tab for the selected SQL connection.
IS_SQLSERVER_SERVER	This property identifies the name of the target server instance (for Microsoft SQL Server and MySQL) or the connect URL string or local net service name (for Oracle). This property is the default value for the Target Server Property Name setting on the Advanced tab for the selected SQL connection.

Table 4-1 • Default SQL Login Properties (cont.)

Property	Description
IS_SQLSERVER_USERNAME	This property identifies the login ID that should be used for server authentication. This property is the default value for the Server Authentication Login ID Property Name setting on the Advanced tab for the selected SQL connection.

If you want to override one of these Windows Installer property for an existing connection, add a new property in the Property Manager. Then, in the SQL Scripts view, select the connection. On the Advanced tab, select the name of the new property in the appropriate list.

If you want to store the values of any of these properties on the target system for later use by your product, you can do so. Following are examples of ways you can use these properties:

- In the Registry view, create a registry value whose data is **[IS_SQLSERVER_SERVER]**. At run time, when your installation creates the registry value, the data for that registry value is set to the name of the SQL catalog.
- Use the Text File Changes view to configure text string replacements that you want to occur at run time. In this view, add a text file reference that describes a file that is installed with your product, and then specify the search-and-replace criteria. For the search-and-replace criteria, you can enter **[IS_SQLSERVER_DATABASE]** in place of the name of the SQL Server machine. At run time, when your installation edits the text file, the name of the SQL Server machine is written in the text file.

Tip • *For a list of additional Windows Installer properties that you can define in your project to override default SQL run-time behavior, see Overriding the Default SQL Run-Time Behavior.*

Project • *In a Basic MSI installation, the built-in SQLLogin dialog lets end users configure the aforementioned properties. If you change any of the SQL properties on the Advanced tab of a SQL connection in a Basic MSI project, the corresponding properties are not automatically updated in the SQLLogin dialog in the Dialogs view. Therefore, you must manually change the properties in the dialog to match the properties that selected on the Advanced tab of the SQL connection.*

Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

InstallShield lets you specify whether new SQL database connections that you add to your project should share the same Windows Installer properties by default.

For example, if you want connections to share the same default Windows Installer properties, you could add one connection to your project in the SQL Scripts view, and specify a catalog name of **MyConnection**. If you add a second connection to your project, InstallShield uses the same catalog name of **MyConnection** for that second connection. If you change the catalog name of either connection, InstallShield automatically updates the catalog name of the other connection, since both are based on the same Windows Installer property.

Task ***To specify whether SQL connections should share the same Windows Installer properties:***

1. On the **Tools** menu, click **Options**. The **Options** dialog box opens.
2. Click the **SQL Scripts** tab.
3. Select or clear the **Generate unique Windows Installer properties for new connections** check box:
   - To share Windows Installer properties between any new connections that you add, clear this check box.
   - To use different Windows Installer properties for any new connections that you add, select this check box.

If you select the check box and then add a second connection to your project, InstallShield creates a new set of Windows Installer properties for the second connection.

If you clear the check box and then add a second connection, InstallShield uses the default Windows Installer properties for the second connection. The default Windows Installer properties are the ones that were added for the first connection that you added to your project.

Tip *• If you want to override a Windows Installer property for an existing connection, add a new property in the Property Manager. Then, in the SQL Scripts view, select the connection. On the Advanced tab, select the name of the new property in the appropriate list.*

Project *• For Basic MSI projects, the built-in SQLLogin dialog is designed to work with the default Windows Installer properties for the first SQL connection that was added to the project. If you select the* ***Generate unique Windows Installer property for new connections*** *option, you need to duplicate the dialog for each connection to work with the new Windows Installer properties.*

Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The InstallScript language includes many built-in SQL run-time (SQLRT) functions that begin with a prefix of *SQLRT*. Some of the functions are available only in InstallScript projects, some are available only in InstallScript MSI projects, and some are available in both project types.

The SQLRT functions require that you configure SQL information through the SQL Scripts view; the configuration information is written to the **SQLRT.ini** file so that the SQL run-time functions work properly. For InstallScript projects, the SQLRT functions are in the **SQLRT.obl** file, and they call the **SQLRT.dll** file. For InstallScript MSI projects, the SQLRT functions are in the **SQLCONV.obl** file, and they call the **ISSQLSRV.dll** file. These support files are added automatically to your project when you use the SQL Scripts view.

The **SQLRTInitialize2** function initializes the SQL server run time. In InstallScript projects, the **SQLRTInitialize2** function is called automatically during the OnSQLServerInitialize event handler. In InstallScript MSI projects, the **SQLRTInitialize2** function is called automatically during the OnSQLLogin event handler. If you need to call one of the SQLRT functions before the OnSQLServerInitialize or OnSQLLogin events, you must first call the **SQLRTInitialize2** function.

***Note** • In earlier versions of InstallShield, the **SQLRTInitialize** function was used to initialize the SQL server run time in InstallScript projects. This function has been superseded by the **SQLRTInitialize2** function.*

Adding a New SQL Connection

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

In the SQL Scripts view, scripts are organized by connection, since no script can run on a server until a connection has been established. Therefore, before you can add any SQL scripts to your project, you must first create a SQL connection.

Task ***To create a new SQL connection:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. Right-click the **SQL Scripts** explorer and click **New SQL Connection**.

InstallShield adds a new connection in the explorer. Use the tabs in the right pane to configure the settings that are associated with this connection.

***Note** • The SQLLogin and SQLBrowse dialogs let end users use alias names for connecting and browsing to SQL Server databases.*

Overriding the Default TCP/IP Network Library with a Different Protocol for a SQL Server Database



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

By default, InstallShield installations use the TCP/IP network library when connecting to a SQL Server database. You can override this default behavior as needed if you want to use a different protocol.



Task ***To override the default TCP/IP network library when connecting to a SQL Server database:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. Right-click the **SQL Scripts** explorer and click **New SQL Connection**. InstallShield adds a new connection in the explorer. Use the tabs in the right pane to configure the settings that are associated with this connection.
3. In the View List under **Additional Tools**, click **Direct Editor**.
4. In the **Tables** explorer, click the **Property** table.
5. Find the IS_SQLSERVER_NETLIB_MS field. The default value is:

   Network Library=DBMSSOCN

   DBMSSOCN refers to the name of the module for the TCP/IP network library, without the file extension.
6. Replace the default DBMSSOCN name as required. For example:
   - To use the Named Pipes network library, specify **DBNMPNTW**.
   - To use SPX/IPX, specify **DBMSSPXN**.
   - To use Banyan Vines, specify **DBMSVINN**.
   - To use Multi-Protocol (Windows RPC), specify **DBMSRPCN**.

At run time, the installation uses the protocol that you specified when connecting to a SQL Server database.

Requirements for Connecting to Instances of SQL Server Express LocalDB



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

If you want your installation to support connections to instances of SQL Server Express LocalDB, the target system must have the SQL Server Native Client 11 ODBC driver. To ensure that it is present on target systems, you can include the Microsoft SQL Server 2012 Native Client prerequisite in your installation. You also must configure your project to use the driver.

Task ***To include the appropriate SQL Server Native Client prerequisite in your project for connecting to instances of SQL Server Express LocalDB:***

1. For Basic MSI and InstallScript MSI projects: In the View List under **Application Data**, click **Redistributables**.

   For InstallScript projects: In the View List under **Application Data**, click **Prerequisites**.
2. In the list of redistributables, select the appropriate **Microsoft SQL Server 2012 Native Client** check box.

The InstallShield prerequisite is launched only if the conditions that are defined in the InstallShield prerequisite file (.prq) are met. To see the list of conditions, click the SQL Server Native Client prerequisite in the Redistributables view or the Prerequisites view, and then review the details that are listed in the details pane on the right. You can hide or show the details pane by clicking the Show Details button in these views.

Task ***To configure your InstallShield installation project so that your installation uses the SQL Server Native Client 11 ODBC driver:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, click the **ISSQLDBMetaData** table.
3. In the **ISSQLDBMetaData** column, find the **MSSQLServer** row.
4. Replace the value of the **AdoDriverName** column with the following:

   sqlncli11

At run time, the installation uses the SQL Server Native Client 11 ODBC driver when connecting to a SQL Server Express LocalDB database.

Adding a New SQL Script

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Task ***To add a new SQL script once you have created a new SQL connection:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. Right-click the new connection and click **New Script**.

When you add a new script to your project, InstallShield adds a new component for the script. You must associate the script with a feature. If one does not exist, the Create a New Feature dialog box opens when you add the SQL script, prompting you to create a new feature. You can change the script-feature association later by using the **Select Features the SQL Script Belongs to** area on the General tab in the SQL Scripts view for the SQL script.

Tip • *You can also add scripts to your project by importing or inserting them. For more information, see Inserting and Importing SQL Scripts.*

Note • *If you have Microsoft SQL Server Management Studio or Microsoft SQL Server 2000 SQL Query Analyzer installed on your system, you can open a new SQL script that you have added to your project to test, edit, and syntax-check the script. To launch one of those tools and open your script from within InstallShield, right-click the script file in the SQL Scripts view and then click Open Script in Microsoft SQL Server Management Studio. InstallShield searches for one of the following tools and launches the first one that it finds:*

- *Microsoft SQL Server 2008 Management Studio (any edition, including Express;* **ssms.exe***)*
- *Microsoft SQL Server 2005 Management Studio (***SqlWb.exe***)*
- *Microsoft SQL Server 2005 Management Studio Express (***ssmsee.exe***)*
- *Microsoft SQL Server 2000 Query Analyzer (***isqlw.exe***)*

Inserting and Importing SQL Scripts

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

InstallShield enables you to reuse SQL script files (.sql) in multiple projects. You can insert or import script files into a project through the SQL Scripts view:

- Inserting a script file creates a link to the script file in its current location.
- Importing a script file copies the script file to the folder containing the script files for your project. The script files that you import can be stored somewhere on your system, or they can be stored in a repository.

Inserting SQL Script Files

Task ***To insert a SQL script file:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, add a SQL connection if you have not already done so.
3. Right-click the SQL connection and then click **Insert Script Files**. The **Open** dialog box opens.

4. Select the SQL script file (.sql) that you want to insert.
5. Click **Open**.

Importing SQL Script Files

Task ***To import a SQL script file:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, add a SQL connection if you have not already done so.
3. Right-click the SQL connection and then click **Import Script Files**. The Import SQL Script Files Dialog Box opens.
4. Do one of the following:
   - In the **Repository Items** box, click the SQL script file (.sql) that you want to add to your project.
   - If the script file that you want to import is not stored in the repository, click the browse button to select it.
5. Click **OK**.

When you insert or import a new script to your project, InstallShield adds a new component for the script. You must associate the script with a feature. If one does not exist, the Create a New Feature dialog box opens when you add the SQL script, prompting you to create a new feature. You can change the script-feature association later by using the **Select Features the SQL Script Belongs to** area on the General tab in the SQL Scripts view for the SQL script.

Importing a SQL Server Database and Generating a SQL Script File

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Task ***To import an existing Microsoft SQL Server database and then generate a script file from it:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. Right-click the **SQL Scripts** explorer and then click **Database Import Wizard**. The **Database Import Wizard** opens.
3. Complete the panels in the Database Import Wizard.

The Database Import Wizard will guide you through the process of importing your database settings and generating a SQL script file based on those settings and other options that you determine.

Note *• The import database functionality applies to the Microsoft SQL Server Database. The script generated by the Database Import Wizard is not compatible with other database types.*

Editing a SQL Script File

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Once you have created, inserted, or imported a script file, you can edit it from within InstallShield.

Project *• In an InstallScript project, if you are working on a script that had overridden OnFirstUIBefore before upgrading to InstallShield and it is not calling OnSQLServerInitialize, you should add that code to your script file.*

Task

To edit a script file in InstallShield:

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, click the script file that you want to edit.
3. Click the **Script** tab, which displays the contents of your script file.
4. Edit the file as needed.

If you want to re-create your script each time that you build your project, you can select the Regenerate SQL Script at Build check box on the Database Import tab for the script in the SQL Scripts view. A backup of any existing script file is always saved first.

Note *• If you have Microsoft SQL Server Management Studio or Microsoft SQL Server 2000 SQL Query Analyzer installed on your system, you can open a new SQL script that you have added to your project to test, edit, and syntax-check the script. To launch one of those tools and open your script from within InstallShield, right-click the script file in the SQL Scripts view and then click Open Script in Microsoft SQL Server Management Studio. InstallShield searches for one of the following tools and launches the first one that it finds:*

- *Microsoft SQL Server 2008 Management Studio (any edition, including Express;* ***ssms.exe****)*
- *Microsoft SQL Server 2005 Management Studio (****SqlWb.exe****)*
- *Microsoft SQL Server 2005 Management Studio Express (****ssmsee.exe****)*
- *Microsoft SQL Server 2000 Query Analyzer (****isqlw.exe****)*

Specifying a Version Number for a SQL Script File

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

You can assign a schema version number to a SQL script file after you have a created or imported a script file in the SQL Scripts view. Specifying schema version information helps you to trigger the launching of the SQL script only when it is appropriate.

Task ***To specify the schema version for your SQL script file:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, click the SQL script that you want to configure.
3. Click the **General** tab.
4. In the **Schema Version (xxxxx.xxxxx.xxxxx.xxxxx)** box, type the schema version number for your SQL script

The installation checks the current schema version that is on the target database. The schema version is stored in the **ISSchema** column of the custom table named **InstallShield**. When you specify a schema version for a SQL script, the installation runs the script only if the script schema version number is greater than the current schema version number. Once the script is executed, the installation updates the current schema version on the target database to reflect the new schema version number.

If you do not specify a schema version number for a SQL script, the script is always launched.

For example, your installation may create a new connection to a database called **TestDB** and then create a script that is called **TestScript** and that has a schema version number of **12345.54321.12345.54321**. The installation creates the custom **InstallShield** table in the **TestDB** database and stores the schema version in the **ISSchema** column of that table. If another installation is run on that system, and this installation also has a SQL script called **TestScript**, the SQL script is executed only if the schema version number of this other script is greater than the version number that is stored in the **ISSchema** column in the **InstallShield** table of the target database. If the script is executed, the installation updates the **ISSchema** column with the new schema version number.

Tip *• When you specify a number for the Schema Version setting and InstallShield adds the custom* ***InstallShield*** *table for storing the schema version number on the target database, the data is not automatically removed upon uninstallation. Therefore, if you want the installation to be able to roll back the changes, you need to create a custom script upon uninstallation to drop the* ***InstallShield*** *table.*

Specifying the Order for Running Multiple SQL Scripts That Are Associated with a Connection

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

If you add more than one SQL script to a SQL connection in your project, you can specify the order in which the installation runs those SQL scripts on the target machine. The procedure for specifying the order differs for Windows Installer–based and InstallScript-based projects.

Specifying the SQL Script Order in Basic MSI, DIM, and InstallScript MSI Projects

The order in which a connection's SQL scripts are listed in the SQL Scripts view of a Basic MSI, DIM, and InstallScript MSI project is the order in which the installations run the SQL scripts. For example, if you create a connection in the SQL Scripts view and add two SQL scripts to that connection, the script that is listed first under the SQL connection is the one that the installation runs first.

Task ***To change the order in which a connection's SQL scripts are run:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, right-click the SQL script that you want to move and click **Move Up** or **Move Down**.

The next time that you build and run the installation, the order is updated.

Specifying the SQL Script Order in InstallScript Projects

By default for InstallScript installations, batch mode for SQL scripts is disabled; therefore, the order in which a connection's SQL scripts are run corresponds with the order in which the installation processes the associated components.

If you want to be able to override this default behavior and specify a particular order for running the SQL scripts, you can do so by enabling the batch mode in the SQL Scripts view.

Task ***To enable or disable batch mode:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. Right-click the **SQL Scripts** explorer and click **Batch Mode**.

Batch mode is enabled if the Batch Mode command has a check mark; it is disabled if the command does not have a check mark.

The **SQLRTGetBatchList** function returns the list of components that are associated with SQL scripts that need to be run when batch mode is enabled. For more information, see SQLRTGetBatchList.

Overriding the Default SQL Run-Time Behavior

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

You can define the following Windows Installer properties to override default run-time behavior:

Table 4-2 • Windows Installer Properties for SQL

Property	Description
IS_SQLSERVER_CONNECTIONS_TO_VALIDATE	Overrides the connections that will be tested when clicking the Next button on the SQLLogin dialog. Specify multiple connections by separating each with a semicolon. By default, all the connections in the **ISSQLConnection** table will be validated.
IS_SQLSERVER_CXNS_ABSENT_FROM_INSTALL	Specifies one or more SQL connections that should be skipped during installation or uninstallation. To specify more than one SQL connection, separate each with a semicolon (;). To skip all of the SQL connections, set the value of this property to **ALL**. Using this property is helpful if you cannot uninstall a product because of a SQL scripting error.
IS_SQLSERVER_DO_NOT_USE_REG	Specifies not to use the stored SQL Server login information written in the registry. Since the SQLLogin dialog is not displayed in maintenance and uninstall modes, the installation stores the login information that is specified during installation. If you do not want this behavior, set the **IS_SQLSERVER_DO_NOT_USE_REG** property.
IS_SQLSERVER_LOCAL_ONLY	Specifies to show only the local SQL Server in the SQL Server browse combo box and list box controls. ***Note*** • *To show only the local SQL Server and either remote SQL Servers or SQL Server aliases, you can set the **IS_SQLSERVER_LOCAL_ONLY** property, as well as either the **IS_SQLSERVER_REMOTE_ONLY** property or the **IS_SQLSERVER_ALIAS_ONLY** property.*

Table 4-2 • Windows Installer Properties for SQL (cont.)

Property	Description
IS_SQLSERVER_REMOTE_ONLY	Specifies to show only the remote SQL Servers in the SQL Server browse combo box and list box controls. ***Note*** *• To show only the remote SQL Servers and either the local SQL Server or SQL Server aliases, you can set the* ***IS_SQLSERVER_REMOTE_ONLY*** *property, as well as either the* ***IS_SQLSERVER_LOCAL_ONLY*** *property or the* ***IS_SQLSERVER_ALIAS_ONLY*** *property.*
IS_SQLSERVER_ALIAS_ONLY	Specify to show only the SQL Server aliases in the SQL Server browse combo box and list box controls. ***Note*** *• To show only the SQL Server aliases and either the local SQL Server or remote SQL Servers, you can set the* ***IS_SQLSERVER_ALIAS_ONLY*** *property, as well either the* ***IS_SQLSERVER_LOCAL_ONLY*** *property or the* ***IS_SQLSERVER_REMOTE_ONLY*** *property.*

Project *• For Basic MSI projects, all connections are linked to the SQLLogin dialog. To display multiple SQLLogin dialogs, you can copy the SQL dialogs from the Dialogs view and modify their behavior and events similar to the default SQL dialogs. Remember to create new properties, and set them in the connection's Advanced tab of the SQL Scripts view. You can use those new properties when you modify the copies of the SQL dialogs that you made.*

Handling SQL Run-Time Errors

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Task ***To set SQL script error-handling properties in the interface:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, click the SQL script file for which you want to add error handling.
3. Click the **Runtime** tab.

4. In the **Script Error Handling** area, select one of the options listed, or click the **Custom** button to override the script's default error handling. Available options are:
   - On Error, Goto Next Script
   - On Error, Goto Next Statement
   - On Error, Abort Installation

Setting Up a Database Server Type Condition for SQL Scripts



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

InstallShield enables you to create a single connection that targets both Microsoft SQL Server and MySQL and that has multiple SQL scripts specific to each database server technology. However, a connection is created based on whichever database type is detected first. Therefore, scripts are run only for the detected database type, and the scripts that are specific to the non-detected database server type fail. For example, if the run time detects a Microsoft SQL Server, then the scripts associated with Microsoft SQL Server run, and the scripts that are specific to MySQL fail.

As a workaround to this behavior, it is recommended that you set a condition in your SQL script so that a SQL scripting error occurs when the installation is running a non-detected database server type script. Then you can set up custom error handling for the scripting error and skip to the next script for the connection. The following instructions discuss how you can set this database server type condition for scripts.



Task ***To set up a database server type condition for a script that targets Microsoft SQL Server:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, click the script file for which you are creating the condition.
3. Click the **Script** tab.
4. Add the following statement at the beginning of the script:

   ```
   SELECT @@ROWCOUNT
   ```

5. Click the **Runtime** tab.
6. In the **Script Error Handling** area, click the **Custom** button. The **Custom Error Handling** dialog box opens.
7. Click the **Click here to add a new item** row.
8. In the **Error Number** column, type **1193**. This is the error number that MySQL returns when it does not have the specified system variable that you added at the beginning of the script.
9. In the **Behavior** column, click **On Error, Goto Next Script**.

10. In the **Project Wide** column, click **No**.
11. Click **OK**.

Task ***To set up a database server type condition for a script that targets MySQL:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, click the script file for which you are creating the condition.
3. Click the **Script** tab.
4. Add the following statement at the beginning of the script:

   ```
   SELECT @@table_cache
   ```
5. Click the **Runtime** tab.
6. In the **Script Error Handling** area, click the **Custom** button. The **Custom Error Handling** dialog box opens.
7. Click the **Click here to add a new item** row.
8. In the **Error Number** column, type **137**. This is the error number that Microsoft SQL Server returns when it does not have the specified system variable that you added at the beginning of the script.
9. In the **Behavior** column, click **On Error, Goto Next Script**.
10. In the **Project Wide** column, click **No**.
11. Click **OK**.

Conditionally Launching a SQL Script in a Windows Installer-Based Installation

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

You may want your installation to launch a SQL script only if certain conditions are met on the target system. For example, your SQL script may require that the end user have administrator rights. If an end user does not have administrator rights, the SQL script is not launched.

Task ***To create a SQL script condition:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, click the script file for which you are creating the condition.
3. Click the **Runtime** tab.
4. In the **Script Condition** area, select the **Specify a Conditional Statement** check box.

5. Click the ellipsis button (**...**). The **Condition Builder** dialog box opens.
6. Create one or more conditions.

The SQL scripts are tied to component states. The condition that is set in the SQL Scripts view is the condition for the SQL script's component. If the component conditions are met on the target system, the installation installs the SQL script. If the installation installs a SQL script that you did not expect to be installed on a target system, generating a log file for your installation may help you determine why the SQL script was installed.



***Tip** • For information on conditions and on condition syntax, see Building Conditional Statements and Conditional Statement Syntax.*

Conditionally Launching a SQL Script in an InstallScript Installation



***Project** • This information applies to InstallScript projects.*

You may want your installation to launch a SQL script only if certain conditions are met on the target system.

InstallShield generates a set of default global event handlers, each of which is a function scripted in the InstallScript language. The following SQL-related events are automatically called by the InstallShield framework:

- OnSQLServerInitialize
- OnSQLComponentInstalled

OnSQLServerInitialize is called by OnFirstUIBefore, and OnSQLComponentInstalled is called during file transfer, for each component installed.



***Note** • If you are working on a script that had overridden OnFirstUIBefore before upgrading to InstallShield and it is not calling OnSQLServerInitialize, you should add that code to your script file.*

In your script, you can modify OnSQLServerInitialize and OnSQLComponentInstalled to perform checks for different things. For example, you can check for a user with administrator rights in the sample code below.

```
function OnSQLComponentInstalled(szComponent)
string sMessage;
string sData;
number nResult;
begin

    if( Is( USER_ADMINISTRATOR, sData ) ) then

        nResult = SQLRTComponentInstall( szComponent );

        if( nResult = SQL_ERROR_ABORT ) then

            sMessage = SdLoadString( IDS_IFX_SQL_ERROR_RUN_FAILED );
            MessageBox( sMessage, MB_OK );
            abort;

        endif;
```

```
    else
        //User does not have administrator rights, so we do not run scripts
    endif;
end;
```



Note • *You can configure the behavior for script failure in the InstallShield interface when you click a SQL script in the SQL Scripts explorer, and then go to the Runtime tab. The Script Error Handling section lets you select one of the following options:*

- *On Error, Go to Next Script*
- *On Error, Go to Next Statement*
- *On Error, Abort Installation*

Requiring that SQL Scripts Be Run Only Against a Full SQL Server



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Installations that include SQL script support allow end users to run the SQL scripts on Microsoft SQL Server Desktop Engine (MSDE) and on SQL Server Express Edition, by default. If you want end users to be able to run SQL scripts on only a full SQL Server, you can use the SQL Scripts view to configure that for any database connection in your installation.



Task ***To prevent your SQL script files from running on target systems that have MSDE or SQL Server Express Edition:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, click the SQL connection that you want to configure.
3. On the **Requirements** tab, clear the **Allow Installation to Microsoft SQL Server Desktop Engine/SQL Server Express** check box.

Using Windows Installer Properties to Dynamically Replace Strings in SQL Scripts



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

For Basic MSI, DIM, and InstallScript MSI projects, you can specify a replacement for text in a SQL script at run time through the use of Windows Installer properties. This enables you to let end users specify information that is then used in the SQL script that is launched on the target system. Windows Installer uses **MsiFormatRecord** to resolve the properties in the SQL script at run time.

Example

The following procedure demonstrates how to create a database at run time using a custom SQL script that contains information that end users enter on the built-in SQL login dialog: the SQLLogin dialog in a Basic MSI installation or the SQLServerSelectLogin2 dialog in an InstallScript MSI installation. Windows Installer properties are used for the database name and its target location.



Task ***To create a database using a SQL script that contains information that end users specify at run time:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. Right-click the **SQL Scripts** explorer and click **New SQL Connection**. InstallShield adds a new SQL connection.
3. Click the SQL connection, and then click the **General** tab.
4. Clear the **Create Catalog If Absent** check box.
5. In the **SQL Scripts** explorer, right-click the new connection and click **New Script**. InstallShield adds a new SQL script to the SQL connection.
6. Click the SQL script, and then click the **Script** tab.
7. In the script pane, enter the following:

```
if not exists(select name from master.dbo.sysdatabases where name = '%DBNAME%')
begin
CREATE DATABASE %DBNAME%
ON
( NAME = %DBNAME%_dat,
    FILENAME = '%DBPATH%%DBNAME%dat.mdf',
    SIZE = 10,
    MAXSIZE = 50,
    FILEGROWTH = 5 )
LOG ON
( NAME = %DBNAME%_log,
    FILENAME = '%DBPATH%%DBNAME%log.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB ) ;
end
GO
```

8. Configure two text replacements for the SQL script:
   a. Click the **Text Replacement** tab, and then click the **Add** button. The **Find and Replace** dialog box opens.
   b. In the **Find What** box, type the following:

      %DBNAME%

   c. In the **Replace What** box, type the following:

      [IS_SQLSERVER_DATABASE]

d. Click **OK**. The **Find and Replace** dialog box closes.

e. Click the **Add** button. The **Find and Replace** dialog box opens.

f. In the **Find What** box, type the following:

%DBPATH%

g. In the **Replace What** box, type the following:

[INSTALLDIR]

h. Click **OK**. The **Find and Replace** dialog box closes.

9. Click the **Runtime** tab.
10. Clear the **Run Script During Install** check box and select the **Run Script During Login** check box.
11. Build your release.

Using InstallScript Text Substitution to Dynamically Replace Strings in SQL Scripts



Project • *This information applies to InstallScript projects.*

For InstallScript projects, you can specify a replacement for text in a SQL script at run time through the use of text substitution string variables. This enables you to let end users specify information that is then used in the SQL script that is launched on the target system. The InstallScript run-time code uses the **TextSubSubstitute** function to replace the string variable with the appropriate value in the SQL script.

Example

The following procedure demonstrates how to create a database at run time using a custom SQL script that contains information that end users enter on the **SQLServerSelectLogin2** dialog. InstallScript text substitution is used for the database name and its target location.



Task ***To create a database using a SQL script that contains information that end users specify at run time:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. Right-click the **SQL Scripts** explorer and click **New SQL Connection**. InstallShield adds a new SQL connection.
3. Click the SQL connection, and then click the **General** tab.
4. Clear the **Create Catalog If Absent** check box.
5. In the **SQL Scripts** explorer, right-click the new connection and click **New Script**. InstallShield adds a new SQL script to the SQL connection.
6. Click the SQL script, and then click the **Script** tab.
7. In the script pane, enter the following:

```
if not exists(select name from master.dbo.sysdatabases where name = '%DBNAME%')
```

```
begin
CREATE DATABASE %DBNAME%
ON
( NAME = %DBNAME%_dat,
    FILENAME = '%DBPATH%\%DBNAME%dat.mdf',
    SIZE = 10,
    MAXSIZE = 50,
    FILEGROWTH = 5 )
LOG ON
( NAME = %DBNAME%_log,
    FILENAME = '%DBPATH%\%DBNAME%log.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB ) ;
end
GO
```

8. Configure two text replacements for the SQL script:

   a. Click the **Text Replacement** tab, and then click the **Add** button. The **Find and Replace** dialog box opens.

   b. In the **Find What** box, type the following:

      %DBNAME%

   c. In the **Replace What** box, type the following:

      <MYDATABASENAME>

   d. Click **OK**. The **Find and Replace** dialog box closes.

   e. Click the **Add** button. The **Find and Replace** dialog box opens.

   f. In the **Find What** box, type the following:

      %DBPATH%

   g. In the **Replace What** box, type the following:

      <TARGETDIR>

   h. Click **OK**. The **Find and Replace** dialog box closes.

9. Click the **Runtime** tab.

10. Clear the **Run Script During Install** check box and select the **Run Script During Login** check box.

11. In the View List under **Behavior and Logic**, click **InstallScript**.

12. Find the dialog code in the OnSQLServerInitialize event for the dialog that should contain the Database Name control, and add a call to the InstallScript function **TextSubSetValue**. For example, if you want the user name to be the name that the end user specifies on the SQLServerSelectLogin2 dialog, you would add a **TextSubSetValue** call as shown in the following lines of code:

```
    // Display login dialog (without connection name)
    // UNCOMMENT OUT TO SWAP DIALOGS
    // nResult = SQLServerSelectLogin2( szConnection, szServer, szUser, szPassword,
bWinLogin, szDB, FALSE, TRUE );

    // Display login dialog (with connection name)
    // COMMENT OUT TO SWAP DIALOGS
```

```
    nResult = SQLServerSelectLogin2( szConnection, szServer, szUser, szPassword, bWinLogin,
szDB, TRUE, TRUE );
    TextSubSetValue ("<MYDATABASENAME>", szDB, FALSE);
```

13. Build your release.

Requiring a SQL Server-Side Installation for a Windows Installer-Based Project



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

One way to configure your installation so that it runs only on SQL Server machines is to perform a system search for registry information, store the result in a property, and then use the property in a condition that you set. The following step-by-step instructions show how to do this.



Task ***To configure a Windows Installer–based installation to run only on SQL Server machines:***

1. In the View List under **Behavior and Logic**, click **System Search**.
2. Right-click the grid and click **Add**. The **System Search Wizard** opens.
3. In the **What do you want to find** panel, click **Registry entry** and click **Next**.
4. In the **How do you want to look for it** panel, do the following:
   a. In the **Registry Root** list, click **HKEY_LOCAL_MACHINE**.
   b. In the **Registry Key** box, type the following text:

      Software\Microsoft\Microsoft SQL Server

   c. In the **Registry Value** box, type the following text:

      InstalledInstances

   d. Click **Next**.
5. In the **What do you want to do with the value** panel, do the following:
   a. In the **Store the value in this property** box, type the following:

      SQLSERVERFOUND

   b. In the **Additional Options** area, select the **Store the value in the property and use the property in an Install Condition** option.
   c. Click **Finish**. The **Condition Builder** opens.
6. Verify the condition, and type a message that you want end users to see if the registry entry is not found on the system. For example, you can type the following message:

Microsoft SQL Server was not found on this machine. This installation was designed to run only on the server machine.

7. Click **OK**.

InstallShield adds an entry to the System Search grid.

Requiring a Server-Side Installation for an InstallScript Project



Project • *This information applies to InstallScript projects.*

One way to enforce a server-side installation in an InstallScript project is to set up your installation project so that it searches for a specific registry key and value and only installs the installation project if the value is found. See the sample code below for an example of how you can do this in your InstallScript project.

```
function OnBegin()
    string sKey, sValue, sData;
    string sMsg;
    number nType, nSize, nResult;
begin

    RegDBSetDefaultRoot( HKEY_LOCAL_MACHINE );
    sKey = "Software\\Microsoft\\Microsoft SQL Server";
    sValue = "InstalledInstances";
    nResult = RegDBGetKeyValueEx( sKey, sValue, nType, sData, nSize );

    if( nResult < 0 ) then

        //SQL Server registry key is missing
        sMsg = "Microsoft SQL Server was not found on this machine.\n" +
               "This installation was designed to run only on the server machine.";
        MessageBox( sMsg, SEVERE );
        abort;

    endif;

end;
```

Publishing SQL Script Files to a Repository



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

If you have an existing SQL script file that you would like to reuse in other projects or share with other users, you can publish it to a repository.

Task ***To publish a SQL script file to a repository:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. In the **SQL Scripts** explorer, right-click the script file that you would like to publish, and then click **Publish Wizard**. The **Publish Wizard** opens.
3. Complete the panels in the Publish Wizard.

After you have imported a script file from a repository into a project, there is no link between the current script file and the existing repository script file. If you make a change to the script file and then republish it to the repository, it does not affect the script file in the project to which it was imported. However, you can reimport the script file from the repository into your project.

Installing the MySQL ODBC Driver

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

If you want to install and launch the MySQL ODBC driver before your installation is run, you can include the InstallShield prerequisite for MySQL Connector 3.51 in your project.

Task ***To install and launch the MySQL ODBC driver before your installation is run:***

1. For Basic MSI and InstallScript MSI projects: In the View List under **Application Data**, click **Redistributables**.

   For InstallScript projects: In the View List under **Application Data**, click **Prerequisites**.
2. In the list of redistributables, select the **MySQL Connector ODBC 3.51** check box.

Tip • *The MySQL Connector ODBC 3.51 prerequisite is available only if you have downloaded the installer and added the InstallShield prerequisite to your system. For detailed instructions, see Including the MySQL Connector ODBC Prerequisite.*

Deleting a SQL Server Database During Uninstallation

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*

Note • *You cannot delete a database to which you are currently connected.*

If you need to remove a SQL database during uninstallation, you can do so through your SQL script. The following procedure demonstrates how to configure your project and SQL script to delete a Microsoft SQL Server database.

Task ***To remove a Microsoft SQL Server database during uninstallation:***

1. In the View List under **Server Configuration**, click **SQL Scripts**.
2. Add a SQL connection.
3. On the **General** tab, in the **Catalog Name** box, type **Master**.

   Master is the name of the system database that exists on all Microsoft SQL Server systems; since you cannot delete a database to which you are currently connected, you can connect to the Master database and then delete a different database to which you are not connected.
4. Enter the authentication information for the server.
5. Add a new SQL script file.
6. Add the following to the SQL script file:

   ```
   DROP DATABASE DatabaseName
   GO
   ```

   DatabaseName is the name of the database that you want to delete.

Tip • *As an alternative to the aforementioned procedure, you can perform the same operation completely in SQL script. To do so, enter the following code in your SQL script:*

```
USE Master
DROP DATABASE DatabaseName
GO
```

DatabaseName is the name of the database that you want to delete.

Connecting to a Microsoft Windows Azure Database Server and Running SQL Scripts

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Target System Requirements for Connecting to a Microsoft Windows Azure SQL Database Server

Connecting to a Microsoft Windows Azure SQL database server requires the SQL Server Native Client 10.0 ODBC driver on target system that are running the installation. To ensure that this driver is available on target systems, you can include the InstallShield prerequisite for Microsoft SQL Server 2008 Native Client 10.00.2531 in your project.

Task ***To include the Microsoft SQL Server 2008 Native Client 10.00.2531 prerequisite in your project:***

1. For Basic MSI and InstallScript MSI projects: In the View List under **Application Data**, click **Redistributables**.

   For InstallScript projects: In the View List under **Application Data**, click **Prerequisites**.
2. In the list of redistributables, select the appropriate **Microsoft SQL Server 2008 Native Client 10.00.2531** check box or check boxes, depending on whether the architecture of target systems is 32 bit or 64 bit.

The InstallShield prerequisite is launched only if the conditions that are defined in the InstallShield prerequisite file (.prq) are met. To see the list of conditions, click the SQL Server Native Client prerequisite in the Redistributables view or the Prerequisites view, and then review the details that are listed in the details pane on the right. You can hide or show the details pane by clicking the Show Details button in these views.

Specifying the User Name for the Windows Azure SQL Database Connection at Run-Time

If you use the SQLLogin dialog in your installation and your installation is targeting a Windows Azure SQL database, end users should use the following format to enter a string in the Login ID box on the SQLLogin dialog:

DatabaseUserName@ServerName

Following is an example:

MyUserName@wbzdh64drd

Connecting to an Instance of Oracle and Running SQL Scripts

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Note *• If you want your installation to download the Oracle 11g Instant Client whenever it needs to be installed on a target system, check with Oracle to ensure that it is allowed. If it is allowed, you must host the download on your own Web site and add its download path to the InstallShield prerequisite. Flexera Software does not make this installation available for download.*

Connecting to an instance of Oracle requires the following elements on the end-user machine that is running the installation:

- Microsoft ODBC for Oracle
- 32-bit version of the Oracle Instant Client software (even on 64-bit target systems)

The latest Microsoft Data Access Components (MDAC) include support for the Microsoft ODBC for Oracle drivers. However, the Microsoft ODBC for Oracle driver requires Oracle Instant Client software to communicate with Oracle database servers. Therefore, you may want to include the Oracle 11g Instant Client prerequisite to help configure the Oracle Instant Client on target machines upon installation. Oracle does not provide an installation for the files; you can do so easily by using the Oracle Instant Client installation project that is available in the InstallShield Program Files folder. For instructions, see Downloading the Oracle Instant Client and Creating an .msi Package for It.

Note that if your SQL script contains Unicode characters, you may want to use the Oracle ODBC Instant Client, since it has support for running SQL scripts that contain Unicode characters, but Microsoft ODBC for Oracle does not. You may want to include an ODBC Instant Client installation with the Oracle 11g Instant Client prerequisite to help configure the Oracle Instant Client on target machines, and to also set up the ODBC Instant Client. For more information, see Downloading the Oracle Basic Instant Client and the Oracle ODBC Instant Client and Creating an .msi Package and InstallShield Prerequisite for Both. If you plan on using the ODBC Instant Client instead of Microsoft ODBC for Oracle, ensure that you configure your project accordingly. For more information, see Running SQL Scripts with Unicode Characters on an Oracle Database Server.

Downloading the Oracle Instant Client and Creating an .msi Package for It



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Before you can add Oracle 11g Instant Client support to your Basic MSI, InstallScript, or InstallScript MSI project, you must download the Oracle Instant Client from Oracle's Web site. Oracle does not provide an installation for the files, so you also need to create an .msi package; you can do so easily by using the Oracle Instant Client installation project that is available in the InstallShield Program Files folder.

Once you have created the .msi package, you can add it to the appropriate folder on your machine, and then add the InstallShield prerequisite that uses this Oracle Instant Client .msi package.



Note *• Note that if your SQL script contains Unicode characters, you may want to use the Oracle ODBC Instant Client, since it has support for running SQL scripts that contain Unicode characters, but Microsoft ODBC for Oracle does not. You may want to include an ODBC Instant Client installation with the Oracle 11g Instant Client prerequisite to help configure the Oracle Instant Client on target machines, and to also set up the ODBC Instant Client. For more information, see Downloading the Oracle Basic Instant Client and the Oracle ODBC Instant Client and Creating an .msi Package and InstallShield Prerequisite for Both.*

Task ***To download the Oracle Instant Client and create an .msi package:***

1. Visit http://www.oracle.com/technetwork/database/features/instant-client/index-097480.html and download the 11.1.0.x version of the Oracle Instant Client (the Basic package for 32-bit Windows platforms). The download file is called **instantclient-basic-win32-11.1.0.7.0.zip**.

   Important • *The Oracle support in InstallShield requires that the target system have the 32-bit version of the Oracle Instant Client, regardless of whether the target system is a 32-bit or 64-bit system.*

2. Unzip the files to the root of your C drive. Unzipping the files adds all of the files to the following location:

   C:\instantclient_11_1

3. Open the Oracle Instant Client installation project in InstallShield. The path for the project is as follows:

   *`InstallShield Program Files Folder`***\Support\Oracle Instant Client\instantclient-win32-11_1.ism**

   Flexera Software created this project for the 11.1.0.7 version of the Oracle Instant Client; however, you can use this same project for more-recent versions of the Oracle 11g Instant Client, such as 11.2.01, for example.

4. The name that is displayed at run time during the Oracle 11g Instant Client installation is **Oracle11g Instant Client 11.1.0.7**; this name is also used for the .msi file. If you want to change this name to something else to reflect a different version number, such as 11.2.0.1 or 11.2.0.x, you can do so:

   a. In the View List under **Installation Information**, click **General Information**.

   b. In the **Product Name** setting, edit the existing text as needed.

5. If you are using a version that was released after 11.1.0.7 and it requires any dependencies, add the files or merge modules to the project.

6. On the toolbar, click the **Build** button.

InstallShield builds an Oracle 11g Instant Client .msi file and adds it to the following directory so that you can include the Oracle 11g Instant Client in your InstallShield projects:

*`InstallShield Program Files Folder`***\SetupPrerequisites\oracle**

Tip • *If you are using a version that was released after 11.1.0.7, open the Oracle 11g Instant Client 11.1.0.7 prerequisite in the InstallShield Prerequisite Editor. Rename the prerequisite to reflect the appropriate version number. In addition, update the conditions to reflect the appropriate version number.*

For instructions on how to add the Oracle 11g Instant Client to installations, see Installing the Oracle 11g Instant Client.

Downloading the Oracle Basic Instant Client and the Oracle ODBC Instant Client and Creating an .msi Package and InstallShield Prerequisite for Both

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Before you can add support for the Oracle 11g Instant Client and the Oracle ODBC Instant Client to your Basic MSI, InstallScript, or InstallScript MSI project, you must download the Oracle Basic Instant Client and the Oracle ODBC Instant Client from Oracle's Web site. Oracle does not provide an installation for the files, so you also need to create an .msi package; you can do so easily by using the Oracle Instant Client installation project that is available in the InstallShield Program Files folder.

Once you have created the .msi package, you can add it to the appropriate folder on your machine, and then add the InstallShield prerequisite that uses this Oracle .msi package.



Task ***To download the Basic Instant Client and the ODCBC Instant Client and create an .msi package:***

1. Visit http://www.oracle.com/technetwork/database/features/instant-client/index-097480.html and download the 11.1.0.x version of the Oracle Instant Client (the Basic package for 32-bit Windows platforms), as well as the 11.1.0.x version of the Instant Client Package for ODBC. The download files are called **instantclient-basic-win32-11.1.0.7.0.zip** and **instantclient-odbc-win32-11.1.0.7.0.zip**.

   Important • *The Oracle support in InstallShield requires that the target system have the 32-bit version of the Basic Instant Client and of the ODBC Instant Client, regardless of whether the target system is a 32-bit or 64-bit system.*

2. Unzip the files to the root of your C drive. Unzipping the files adds all of the files to the following location:

   C:\instantclient_11_1

   Ensure that the files are not unzipped to **C:\instantclient11_1\instantclient11_1**.

3. Make a copy of the Oracle Instant Client installation project. The path for the project is as follows:

   *`InstallShield Program Files Folder`***\Support\Oracle Instant Client\instantclient-win32-11_1.ism**

   Name the copy **instantclient-win32-odbc11_1.ism**, and put it in the same directory as the original .ism file. The path for this new file is:

   *`InstallShield Program Files Folder`***\Support\Oracle Instant Client\instantclient-win32-odbc11_1.ism**

   Flexera Software created the **instantclient-win32-11_1.ism** project for the 11.1.0.7 version of the Oracle Instant Client; however, you can use this same project for more-recent versions of the Oracle 11g Instant Client and ODBC Instant Client, such as 11.2.01, for example.

4. Open the new project that you created in InstallShield.

5. Update the product code, the upgrade code, and the product name:

   a. In the View List under **Installation Information**, click **General Information**.

   b. In the **Product Code** setting, click the **Generate a new GUID** button (**{..}**).

   c. In the **Upgrade Code** setting, click the **Generate a new GUID** button (**{..}**).

   d. In the **Product Name** setting, change the existing text as needed.

      Oracle11g Instant Client - ODBC 11.1.0.7

The name that you enter is displayed at run time during the Oracle installation. If you want to change this name to something else to reflect a different version number, such as 11.2.0.1 or 11.2.0.x, you can do so.

6. If you are using a version that was released after 11.1.0.7 and it requires any dependencies, add the files or merge modules to the project.
7. Add a custom action that launches the ODBC Instant Client installation:
   a. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
   b. In the center pane, right-click the **Custom Actions** explorer, point to **New EXE**, and click **Path referencing a directory**. InstallShield adds a new custom action.
   c. Configure the custom action's settings in the right pane as follows:
      - Working Directory: INSTALLDIR
      - Filename & Command Line: [INSTALLDIR]odbc_install.exe
      - In-Script Execution: Deferred Execution in System Context
      - Install Exec Sequence: After InstallODBC
      - Install Exec Condition: Not Installed
8. Add a custom action that uninstalls the ODBC Instant Client when end users uninstall the Instant Client:
   a. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
   b. In the center pane, right-click the **Custom Actions** explorer, point to **New EXE**, and click **Path referencing a directory**. InstallShield adds a new custom action.
   c. Configure the custom action's settings in the right pane as follows:
      - Working Directory: INSTALLDIR
      - Filename & Command Line: [INSTALLDIR]odbc_uninstall.exe
      - In-Script Execution: Deferred Execution in System Context
      - Install Exec Sequence: After RemoveODBC
      - Install Exec Condition: REMOVE="ALL"
9. Specify a name for the .msi package file:
   a. In the View List under **Media**, click **Releases**.
   b. In the center pane, in the **Releases** explorer, click the product configuration called **Product Configuration 1**.
   c. In the **MSI Package File Name** setting, enter the following name:

      Oracle11g Instant Client - ODBC 11.1

      InstallShield will use the name that you enter for the .msi package that it creates for the Basic Instant Client and ODBC Instant Client. This is the file that will be launched by the InstallShield prerequisite that you create.
10. On the toolbar, click the **Build** button.

InstallShield builds an Oracle .msi file and adds it to the following directory so that you can include the **Oracle11g Instant Client - ODBC 11.1.msi** file in your InstallShield projects:

*InstallShield Program Files Folder***SetupPrerequisites\oracle**



Task ***To create an InstallShield prerequisite that launches the .msi package that you created in the aforementioned procedure:***

1. Make a copy of the existing InstallShield prerequisite (.prq) for the Oracle11g Instant Client. The path for the prerequisite is as follows:

   *InstallShield Program Files Folder***SetupPrerequisites\Oracle11g Instant Client 11.1.0.7.prq**

   Name the copy **Oracle11g Instant Client - ODBC 11.1.0.7.prq**, and put it in the same directory as the original .prq file. The path for this new file is:

   *InstallShield Program Files Folder***SetupPrerequisites\Oracle11g Instant Client - ODBC 11.1.0.7.prq**

2. On the **Tools** menu, click **Prerequisite Editor**. The **InstallShield Prerequisite Editor** opens.
3. On the **File** menu, click **Open**. The **Open** dialog box opens.
4. Browse to the new **Oracle11g Instant Client - ODBC 11.1.0.7.prq** file, and then click the **Open** button.
5. On the **Properties** tab, in the **Unique identifier for InstallShield prerequisite** setting, enter:

   Oracle11g Instant Client – ODBC 11.1.0.7

   If you are using a different version, update the identifier accordingly.

6. Configure the conditions for the InstallShield prerequisite:
   a. Click the **Conditions** tab.
   b. Click each existing condition and then click the **Remove** button.
   c. Click the **Add** button. The **Prerequisite Condition** dialog box opens.
   d. Select the **A registry key does or does not exist** option.
   e. In the **Specify the registry key name to check** setting, enter:

      HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\Oracle in instantclient11_1

      If you are using a different version, configure the condition accordingly.

   f. Select the **32-bit** option.
   g. Select the **If the specified registry key DOES NOT EXIST** option, and then click **OK**.
7. Specify the appropriate .msi file to include in your InstallShield prerequisite:
   a. Click the **Files to Include** tab.
   b. Select the existing **Oracle11g Instant Client 11.1.msi** file, and then click the **Modify** button. The **New File** dialog box opens.
   c. Select the **Oracle11g Instant Client - ODBC 11.1.msi** file that you built in the aforementioned procedure, and then click **OK**.

8. Specify the file that the InstallShield prerequisite should launch:
   a. Click the **Application to Run** tab.
   b. In the **Specify the application you wish to launch** setting, select the .msi file.
9. On the **File** menu, click **Save**.

For instructions on how to add the Oracle 11g Instant Client and the ODBC Instant Client to installations, see Installing the Oracle 11g Instant Client.

If you plan on using the Oracle ODBC Instant Client instead of Microsoft ODBC for Oracle, ensure that you configure your project accordingly. For more information, see Running SQL Scripts with Unicode Characters on an Oracle Database Server.

Installing the Oracle 11g Instant Client



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

If you want to install the Oracle 11g Instant Client when your installation is run, you can include the Oracle 11g Instant Client prerequisite in your project.

Note that before you can include the InstallShield prerequisite, you must download the Oracle Basic Instant Client and create an .msi package. If you also want the Oracle ODBC Instant Client to be installed—for example, if you need to use Unicode support with Oracle—you must download the Basic Instant Client and the ODBC Instant Client, build an .msi package, and create an InstallShield prerequisite; you can use the Oracle 11g Instant Client prerequisite as a template, and modify the settings as needed. For more information, see:

- Downloading the Oracle Instant Client and Creating an .msi Package for It
- Downloading the Oracle Basic Instant Client and the Oracle ODBC Instant Client and Creating an .msi Package and InstallShield Prerequisite for Both



Important • *The Oracle support in InstallShield requires that the target system have the 32-bit version of the Oracle Instant Client, regardless of whether the target system is a 32-bit or 64-bit system.*



Task ***To include the Oracle 11g Instant Client prerequisite in your project:***

1. For Basic MSI and InstallScript MSI projects: In the View List under **Application Data**, click **Redistributables**.

   For InstallScript projects: In the View List under **Application Data**, click **Prerequisites**.
2. In the list of redistributables, select the **Oracle11g Instant Client 11.1.0.7** check box (or the check box for whatever Oracle prerequisite that you are using).

The InstallShield prerequisite is launched only if the conditions that are defined in the InstallShield prerequisite file (.prq) are met. To see the list of conditions, click the Oracle prerequisite in the Redistributables view or the Prerequisites view, and then review the details that are listed in the details pane on the right. You can hide or show the details pane by clicking the Show Details button in these views.



Note • *If you want your installation to download the Oracle 11g Instant Client whenever it needs to be installed on a target system, check with Oracle to ensure that it is allowed. If it is allowed, you must host the download on your own Web site and add its download path to the InstallShield prerequisite. Flexera Software does not make this installation available for download.*

Running SQL Scripts with Unicode Characters on an Oracle Database Server



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

For installations that use the default SQL script support that is available in the SQL Scripts view of InstallShield, Microsoft ODBC for Oracle is used to connect to an Oracle database server on a target system. In some cases, you may want to override the default behavior, and use the Oracle ODBC Instant Client instead of Microsoft ODBC for Oracle.

For example, if your SQL script contains Unicode characters, you may want to use the Oracle ODBC Instant Client, since it has support for running SQL scripts that contain Unicode characters, but Microsoft ODBC for Oracle does not.



Task ***To configure your InstallShield project so that your installation uses the Oracle ODBC Instant Client instead of Microsoft ODBC for Oracle:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, click the **ISSQLDBMetaData** table.
3. Perform the following steps for the **Oracle** record:
   a. Change the AdoDriverName field to the following value:

      {Oracle in instantclient11_1}

   b. Change the AdoCxnServer field to the following value:

      DBQ=

At run time, the installation uses the Oracle ODBC Instant Client for connecting to the Oracle database and running SQL scripts.

Tip • *The Oracle ODBC Instant Client is available on a target system if the ODBC package of the Oracle Instant Client software is installed. To learn how to add it to your installation, see Downloading the Oracle Basic Instant Client and the Oracle ODBC Instant Client and Creating an .msi Package and InstallShield Prerequisite for Both.*

Creating a Sample Installation that Creates a SQL Server Database by Running Customized SQL Script

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The following procedure demonstrates how to create an installation that creates a SQL Server database through customized SQL script.

Task ***To create an installation that creates a SQL Server database on the target machine by running customized SQL script:***

1. Create a new Basic MSI or InstallScript MSI project.
2. In the View List under **Behavior and Logic**, click **Property Manager**.
3. Create a new property that has the following name:

   IS_SQLSERVER_DATABASE2
4. In the View List under **Server Configuration**, click **SQL Scripts**.
5. Add and configure a new SQL connection:
   a. Right-click the **SQL Scripts** explorer and click **New SQL Connection**. InstallShield adds a new connection with the name **NewConnection1** as the default name.
   b. In the **SQL Scripts** explorer, click the **NewConnection1** item, and then click the **General** tab.
   c. In the **Default Target Server Name (optional)** box, type **TESTSQLSERVER**.
   d. Clear the **Create Catalog If Absent** check box.
   e. In the **Connect using** area, select the **Server authentication using the Login ID and password below** option.
   f. In the **Login ID** box, type **sa**. Leave the **Password** box blank.
   g. Click the **Requirements** tab.
   h. Ensure that the **Microsoft SQL Server** check box is selected and the **MySQL** and **Oracle** check boxes are cleared.
6. Add and configure a new SQL script for NewConnection1:
   a. In the **SQL Scripts** explorer, right-click **NewConnection1** and click **New Script**.

b. Change the name of the script to **NewScript1**.

c. In the **SQL Scripts** explorer, click **NewScript1**, and then click the **Script** tab.

d. In the script editor pane, add the following script:

```
CREATE DATABASE [TestDB] ON (NAME = N' TestDB', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL\data\testdb.mdf' , SIZE = 3, FILEGROWTH = 10%) LOG ON (NAME = N' TestDB_log',
FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL\data\testdb.ldf' , SIZE = 1, FILEGROWTH
= 10%)
COLLATE SQL_Latin1_General_CP1_CI_AS
```

e. Click the **Runtime** tab.

f. In the **Script Execution** area, select the **Run Script During Login** check box and ensure that the **Run Script During Install** and **Run Script During Uninstall** check boxes are cleared.

7. Add and configure a second new SQL connection:

a. Right-click the **SQL Scripts** explorer and click **New SQL Connection**. InstallShield adds a new connection with the name **NewConnection2** as the default name.

b. In the **SQL Scripts** explorer, click the **NewConnection2** item, and then click the **Advanced** tab.

c. In the **Target Catalog Property Name** list, select **IS_SQLSERVER_DATABASE2**.

d. Click the **General** tab.

e. In the **Catalog Name** box, type **TestDB**.

f. Clear the **Create Catalog If Absent** check box.

g. In the **Default Target Server Name (optional)** box, type **TESTSQLSERVER**.

h. In the **Connect using** area, select the **Server authentication using the Login ID and password below** option.

i. In the **Login ID** box, type **sa**. Leave the **Password** box blank.

j. Click the **Requirements** tab.

k. Ensure that the **Microsoft SQL Server** check box is selected and the **MySQL** and **Oracle** check boxes are cleared.

8. Add and configure a new SQL script for NewConnection2:

a. In the **SQL Scripts** explorer, right-click **NewConnection2** and click **New Script**.

b. Change the name of the script to **NewScript2**.

c. In the **SQL Scripts** explorer, click **NewScript2**, and then click the **Script** tab.

d. In the script editor pane, add the following script:

```
CREATE TABLE TestTable (TestColumn1 CHAR NOT NULL PRIMARY KEY)
```

e. Click the **Runtime** tab.

f. In the **Script Execution** area, select the **Run Script During Install** check box and ensure that the **Run Script During Login** and **Run Script During Uninstall** check boxes are cleared.

When you run the installation, it creates the **TestDB** database and adds a table called **TestTable** to that database.



Project *• You can achieve the same outcome when following the above procedure but using a DIM project, and then adding the DIM to an installation project.*

Creating a Sample Installation that Creates an Oracle Schema by Running Customized SQL Script



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The following procedure demonstrates how to create an installation that creates an Oracle schema through customized SQL script.



Task ***To create an installation that creates an Oracle schema on the target machine by running customized SQL script:***

1. Create a new Basic MSI or InstallScript MSI project.
2. In the View List under **Behavior and Logic**, click **Property Manager**.
3. Create three new properties that have the following names:

   IS_SQLSERVER_DATABASE2

   IS_SQLSERVER_USERNAME2

   IS_SQLSERVER_PASSWORD2
4. In the View List under **Server Configuration**, click **SQL Scripts**.
5. Add and configure a new SQL connection:
   a. Right-click the **SQL Scripts** explorer and click **New SQL Connection**. InstallShield adds a new connection with the name **NewConnection1** as the default name.
   b. In the **SQL Scripts** explorer, click the **NewConnection1** item, and then click the **General** tab.
   c. In the **Default Target Server Name (optional)** box, type the following:

   ```
   //sch01jsmith.installshield.com:1521/orcl
   ```

   d. Clear the **Create Catalog If Absent** check box.
   e. In the **Connect using** area, select the **Server authentication using the Login ID and password below** option.
   f. In the **Login ID** box, type **scott**.
   g. In the **Password** box, type **scott**.
   h. Click the **Requirements** tab.

i. Ensure that the **Oracle** check box is selected and the **Microsoft SQL Server** and **MySQL** check boxes are cleared.

6. Add and configure a new SQL script for NewConnection1:

   a. In the **SQL Scripts** explorer, right-click **NewConnection1** and click **New Script**.

   b. Change the name of the script to **NewScript1**.

   c. In the **SQL Scripts** explorer, click **NewScript1**, and then click the **Script** tab.

   d. In the script editor pane, add the following script:

```
CREATE TABLESPACE TEST_TS LOGGING DATAFILE 'test01.dbf' SIZE 1M AUTOEXTEND ON NEXT 2M MAXSIZE
UNLIMITED
Go
CREATE USER TEST_USER IDENTIFIED BY MYPSWD DEFAULT TABLESPACE TEST_TS QUOTA UNLIMITED on TEST_TS
Go
GRANT CONNECT TO TEST_USER
Go
GRANT DBA TO TEST_USER
Go
ALTER USER TEST_USER DEFAULT ROLE ALL
Go
```

   e. Click the **Runtime** tab.

   f. In the **Script Execution** area, select the **Run Script During Login** check box and ensure that the **Run Script During Install** and **Run Script During Uninstall** check boxes are cleared.

7. Add and configure a second new SQL connection:

   a. Right-click the **SQL Scripts** explorer and click **New SQL Connection**. InstallShield adds a new connection with the name **NewConnection2** as the default name.

   b. In the **SQL Scripts** explorer, click the **NewConnection2** item, and then click the **Advanced** tab.

   c. In the **Target Catalog Property Name** list, select **IS_SQLSERVER_DATABASE2**.

   d. In the **Server Authentication Login ID Property Name** list, select **IS_SQLSERVER_USERNAME2**.

   e. In the **Server Authentication Password Property Name** list, select **IS_SQLSERVER_PASSWORD2**.

   f. Click the **General** tab.

   g. In the **Catalog Name** box, type **TEST_USER**.

   h. Clear the **Create Catalog If Absent** check box.

   i. In the **Default Target Server Name (optional)** box, type the following:

```
//sch01jsmith.installshield.com:1521/orcl
```

   j. In the **Connect using** area, select the **Server authentication using the Login ID and password below** option.

   k. In the **Login ID** box, type **TEST_USER**.

   l. In the **Password** box, type **MYPSWD**.

   m. Click the **Requirements** tab.

n. Ensure that the **Oracle** check box is selected and the **Microsoft SQL Server** and **MySQL** check boxes are cleared.

8. Add and configure a new SQL script for NewConnection2:

   a. In the **SQL Scripts** explorer, right-click **NewConnection2** and click **New Script**.

   b. Change the name of the script to **NewScript2**.

   c. In the **SQL Scripts** explorer, click **NewScript2**, and then click the **Script** tab.

   d. In the script editor pane, add the following script:

   ```
   CREATE TABLE TestTable (TestColumn1 CHAR NOT NULL PRIMARY KEY)
   ```

   e. Click the **Runtime** tab.

   f. In the **Script Execution** area, select the **Run Script During Install** check box and ensure that the **Run Script During Login** and **Run Script During Uninstall** check boxes are cleared.

When you run the installation, it creates the **TEST_USER** user and a **TestTable** table on the **TEST_USER** schema.



Project *• You can achieve the same outcome when following the above procedure but using a DIM project, and then adding the DIM to an installation project.*

Adding a SQLLogin Predefined Wizard Page in a Suite/Advanced UI Project



Project *• This information applies to the Suite/Advanced UI project type.*



Edition *• The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

SQL servers are integral to many applications, especially those that benefit from the multiple package support provided by InstallShield Suite installations. Previously, InstallShield SQL support was limited to Basic MSI, InstallScript, and InstallScript MSI projects. Now, SQL support has been added to Suite/Advanced UI projects, giving you the ability to:

- Add a new SQLLogin predefined wizard page
- Execute SQL statements directly from Suites

The SQLLogin wizard page lets end users enter database server login information (database server name, authentication credentials, database catalog name, etc.) in order to establish a connection to the database server that is targeted by one or more .msi packages in the suite.

Adding a SQLLogin wizard page to your project allows you to:

- Specify properties that identify the SQL login settings in the Suite project and then select the .msi package that receives these properties

- Specify the properties that identify the SQL login settings in the .msi package
- Choose the database technology (Microsoft SQL Server, Microsoft Windows Azure, MySQL, or Oracle) and select the ODBC driver to be targeted
- Ensure that updated SQL drivers will be scheduled for installation via an action in OnBegin

In addition, Suite/Advanced UI projects now support directly executing SQL statements on SQL database servers from the user interface, helping to examine SQL database servers before proceeding the installation. The SQL query result can then be accessed in a Suite property.

The following shows how to add a SQLLogin predefined wizard page to your project. Alternatively, you can manually configure SQL support in your Suite project in order to customize the UI based upon your installation's needs.



Task ***To add a SQLLogin predefined wizard page and optionally execute a SQL statement directly in a Suite/Advanced UI project:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, right-click **Wizard Pages** and then click **Add Predefined Page**. The **User Interface Wizard** opens.
3. In the **Task page** list, select **Enter login information for a database server**.
4. In the **Subsequent page** list, indicate where in the wizard page sequence you want to schedule the database server wizard page. The page that you select in this list will be displayed immediately after the new database server wizard page at run time.
5. In the Task Configuration panels in the User Interface Wizard:
   a. Specify properties that identify the SQL login settings in the Suite project.

   Note *• A suite project may have multiple packages that require different SQL servers or they may need to share.*

   b. Select the .msi package that receives the properties.

   c. Choose the database technology and driver.

   InstallShield adds corresponding Page Entered settings for the Task Configuration properties you specify.
6. To optionally add a SQL statement to be directly executed from the Suite/Advanced UI project at runtime, add a **Run a SQL String** action to your project and specify its subsettings as needed. The SQL query result will be accessible from the Suite property that you specify in the **Data Property** subsetting.

Managing COM+ Applications and Components



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript MSI*
- *Merge Module*

The Component Services view enables you to manage COM+ applications and components for your installation package. You can manage both COM+ server applications and application proxies. A COM+ application proxy consists of a subset of the attributes of the server application, and it enables remote access from a client machine to the machine where the application resides.

Note the following information regarding component services in InstallShield:

- Only non-system COM+ applications can be added to your project. Therefore, InstallShield displays only non-system COM+ applications under the COM+ Applications explorer in the Component Services view.
- Only the COM+ applications that are installed on the local machine are included in the Component Services view and available for you to add to your projects.
- An application proxy is available for COM+ server applications only, not for library applications.

The look and feel of the Component Services view is similar to that of the Component Services administrative tool in the Control Panel.

Managing COM+ Server Applications



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*



Task ***To configure the installation settings for a COM+ server application:***

1. In the View List under **Server Configuration**, click **Component Services**.
2. In the **Component Services** explorer, select the COM+ application that you would like to configure if you have not already done so.
3. On the **Installation** tab, select the **Server** check box.

   Note *• You must select the* ***Proxy*** *check box, the* ***Server*** *check box, or both check boxes. Once you clear one of these check boxes, the other check box remains selected but it is no longer available; this prevents you from clearing both check boxes.*
4. Select the **Refresh the COM+ settings from the client machine at build** check box if appropriate.
5. If your installation does not include the COM+ application proxy support, clear the **Proxy** check box

When you add a COM+ server application to your project, InstallShield creates a corresponding component for each of the features that are associated with the proxy component. This component has the COM+ application's server .dll files.

Note • *If the selected COM+ application includes both server and proxy installations, you can add installation conditions so that the appropriate COM+ application is installed on the target machine. For more information, see Including a COM+ Application that Targets Servers and Client Machines.*

Managing COM+ Application Proxies

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The COM+ application proxy support configures the system settings that enable remote access from a client machine to the machine where the server application resides. You may also need to add the client application to your installation.

Task ***To configure the installation settings for a COM+ application proxy:***

1. In the View List under **Server Configuration**, click **Component Services**.
2. In the **Component Services** explorer, select the COM+ application that you would like to configure if you have not already done so.
3. On the **Installation** tab, select the **Proxy** check box.
4. Clear the **Server** check box if your installation does not include the COM+ server application.

   Note • *You must select the **Proxy** check box, the **Server** check box, or both check boxes. Once you clear one of these check boxes, the other check box remains selected but it is no longer available; this prevents you from clearing both check boxes.*
5. In the **Remote Server Name** box, specify the name of the remote server computer where the application resides. You can type the exact name or use the default **[REMOTESERVERNAME]** property, which is automatically created when you select the **Proxy** check box for a COM+ application in your installation.

   Note • *The default value for the **[REMOTESERVERNAME]** property is the name of the machine used to add the COM+ application to the installation project in InstallShield. To change the value of the **[REMOTESERVERNAME]** property, use the Property Manager view.*

   *If you would like the end user to be able to specify the remote server, add a Remote Server edit field control to an end-user dialog in the **Dialogs** view. Set the **Property** value of this control to* **REMOTESERVERNAME**.

6. Select the **Enable distributed COM on the client machine** check box if appropriate. Clear this check box if you know that distributed component object model (DCOM) is already enabled on all client machines and you will not have administrative privileges on them.

   If you select this check box, Y is written at installation time to the EnableDCOM entry of the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Ole registry key to enable DCOM.

   ***Note** • End users can enable or disable DCOM on their machines using the Component Services administrative tool in the Control Panel. However, the application proxy will not work on a client machine if DCOM is disabled on that machine. For this reason, you may want to select the Enable distributed COM on the client machine check box.*

   If an end user uninstalls the application proxy support, the EnableDCOM registry entry is not changed, even if the installation process involved changing this registry entry to Y on the target machine.

When you add a COM+ application proxy to your project, InstallShield creates a corresponding component for each of the features that are associated with the server component. This component has the COM+ application's server .dll files. These server files are installed on the client machine for the type libraries.



***Note** • If the selected COM+ application includes both server and proxy installations, you can add installation conditions so that the appropriate COM+ application is installed on the target machine. For more information, see Including a COM+ Application that Targets Servers and Client Machines.*

Including a COM+ Application that Targets Servers and Client Machines



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

If you want your installation to install the server application on the server as well as the application proxy support on client machines, you can create installation conditions so that the appropriate component is installed on the target machine. The following example procedure illustrates how you can do this.



Task ***To include a COM+ application in an installation that targets both servers and client machines:***

1. In the **Property Manager** view, create a new property called **COMPLUSTYPE**.
2. In the **Dialogs** view, add a radio button group control to a new or existing end-user dialog, and set the **Property** value of this control to **COMPLUSTYPE**. This radio button group will enable end users to specify which installation they would like to run: server or proxy.
3. Add a radio button for the **Server** option to the radio button group, and set the **Value** property of this control to Server.

4. Add a radio button for the **Proxy** option to the radio button group, and set the **Value** property of this control to Proxy.
5. In the **Component Services** view, select the COM+ application that you would like to configure.
6. On the **Installation** tab, set the parameters for server installations and application proxies.
7. In the **Condition** box under the **Server** check box, type the following:

   COMPLUSTYPE="Server"
8. In the **Condition** box under the **Proxy** check box, type the following:

   COMPLUSTYPE="Proxy"

Managing Internet Information Services



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Internet Information Services (IIS) is a Web server developed by Microsoft. It provides a secure platform for building and deploying Web-based applications, managing Web sites, and publishing information to the Internet or an intranet.

The Internet Information Services view in InstallShield enables you to create and manage new IIS Web sites, applications, virtual directories, application pools, and Web service extensions.

Version-Specific Information for IIS Support in InstallShield



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Following is information about specific versions of IIS:

- IIS is included with Windows 2000 Server and later and Windows XP and later systems. IIS 6 is available only on Windows Server 2003 systems. IIS 7 is available on Windows Vista and Windows Server 2008 systems. IIS 7.5 is available on Windows 7 and Windows Server 2008 R2 systems. IIS is not installed automatically by default.

- Some of the Web site and virtual directory settings in the Internet Information Services view apply to specific versions of IIS. Version-specific information is noted for these settings in the inline help panes in InstallShield. If you configure a version-specific property but a target system does not have the corresponding version of IIS, IIS ignores the version-specific property.

  For example, IIS 7 and IIS 6 do not support the Application Protection property for applications or virtual directories. In the Internet Information Services view, this property is configured through a setting in the Application Settings area for an application or a virtual directory. When you select this setting in the Internet Information Services view, the help pane that is displayed in the lower-right corner indicates that the setting does not apply to IIS 6 or later. If you configure this setting and an end user installs your product on a target system that has IIS 6 or later, the Application Protection setting is ignored.

- Web service extensions, application pools, and all of the associated properties are available only on machines with IIS 6 or later.

  On systems with IIS 7, Web service extensions require that the IIS Metabase and IIS 6 Configuration Compatibility feature be installed. If it is not installed, the installation's log file may show error -2147467259 to inform you that this feature may be required.

  For information on how to determine whether a Windows Vista or Windows Server 2008 system has the IIS Metabase and IIS 6 Configuration Compatibility feature installed, see Determining If a Target System Has IIS 6 or Earlier or the IIS 6 Metabase Compatibility Feature.

- Windows Vista and later and Windows Server 2008 and later systems store settings for individual Web sites, applications, and virtual directories in configuration files that are located at the physical path that you specify in your installation project—in the Content Source Path (Local or UNC) setting in the Internet Information Services view. Therefore, each Web site, application, or virtual directory should have a unique physical path. Otherwise, you may notice unexpected behavior, especially in testing environments, if you have two different applications or virtual directories with the same physical path.

  For example, if you have two virtual directories that have the same physical path, and directory browsing is enabled in one but not the other, the directory browsing setting for the second virtual directory that is created overrides the setting for the first virtual directory.

- On systems that have Windows Server 2003, if IIS 6 is not installed, other IIS directories and sites are still created. IIS 6-specific settings are skipped.

- IIS 5.1 for Windows XP Professional can service only one Web site at a time. This is a limitation of IIS 5.1.

- InstallShield supports version 5 and later of IIS.

Determining Which Version of IIS Is Installed



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

When you add a Web site in the Internet Information Services view, an entry is added to the System Search view to search for the version of IIS that is installed on the target system. The IIS_VERSION property is set by entries in the **RegLocator** and **AppSearch** tables (available in the Direct Editor), which determine the version of IIS that is installed.

The **IIS_VERSION** property contains the IIS version number. If you need to determine the IIS version that is installed, check the value of this property.

Run-Time Requirements for IIS Support



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

IIS support in InstallShield installations works only if IIS is installed on the target machine and the end user has administrative privileges.

During the installation of a package that includes IIS settings, an InstallShield installation checks for the existence of IIS on the target machine. If IIS is not installed, the installation displays a dialog informing the end user that they do not have IIS installed. The dialog gives the end user the option to abort, retry, or ignore:

- If the end user chooses to abort, the installation stops.
- If the end user installs IIS and then chooses to retry, the installation checks for IIS again, and continues with the installation. If the end user does not install IIS but still chooses to retry, the installation checks for IIS again and then displays the dialog again.
- If the end user chooses to ignore, the installation continues, but IIS Web sites, applications, virtual directories, and other IIS elements are not configured.



Note *• InstallShield does not support the creation of Web sites on target machines other than the one on which the installation is running.*



Project *• If you are creating an application pool in an InstallScript project and you want the application pool to be associated with a virtual directory, application, or Web site, you must create the application pool and the virtual directory, application, or Web site within the same component. If you do this, the application pool is created before the virtual directory, application, or Web site is created. This is a requirement of IIS 6 and later, and it is not a limitation for Basic MSI, DIM, InstallScript MSI, or Merge Module projects.*

Specifying Whether a Web Server Should Allow the CMD Command to Be Used for SSI #exec Directives



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Server-side include (SSI) directives instruct a Web server to insert content into a Web page. The #exec type of directive enables the Web server to include the output of a shell command in a Web page.

You can configure an IIS Web server to prevent the CMD command for the #exec directive from being used to execute shell commands, or you can configure it to allow the CMD command to be used to execute this type of command. The SSIEnableCmdDirective registry value for the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters registry key is what determines whether the CMD command is permitted.

InstallShield lets you specify how your installation should configure the SSIEnableCmdDirective registry value on target systems. If you do not want your installation to change the SSIEnableCmdDirective registry value, you can also specify that.

Because of security concerns, the default SSIEnableCmdDirective registry value is FALSE (0); the FALSE (0) value prevents end users from running unauthorized server-side executable files.



Task ***To specify whether a Web server should allow the CMD command to be used for SSI #exec directives:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the center pane, click the **Web Sites** explorer. InstallShield displays the Web server settings in the right pane.
3. For the **SSIEnableCmdDirective registry value** setting, select the appropriate option:
   - **Ignore**—Do not change the SSIEnableCmdDirective registry value on the target system. This is the default option.
   - **FALSE (0)**—Set the SSIEnableCmdDirective registry value on the target system to 0. This prevents the #exec CMD directive of server-side includes to be used to execute shell commands. Note that if you select this value and an IIS Web server has applications that rely on #exec CMD directives, those applications may stop working properly after your installation project's Web site and virtual directory are installed.
   - **TRUE (1)**—Set the SSIEnableCmdDirective registry value on the target system to 1. This allows the #exec CMD directive of server-side includes to be used to execute shell commands.

If you select the FALSE or TRUE options, InstallShield stores the value—either 0 for FALSE or 1 for TRUE—in the **INSTALLSHIELD_SSI_PROP** property.

If one or more Web sites, virtual directories, application pools, or Web service extensions in your installation are installed on a target system and you selected the FALSE or TRUE options for the **SSIEnableCmdDirective registry value** setting, the SSIEnableCmdDirective registry value is updated on the target system.

Note • *If your product is uninstalled from a target system, the SSIEnableCmdDirective registry value is not changed, even if its value was changed during installation.*

Creating a Web Site and Adding an Application or a Virtual Directory

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

The Internet Information Services view in InstallShield is where you add an IIS Web site to your project. It is also where you can add applications and virtual directories to a Web site. Note that you can add a Web site without any applications or virtual directories if you associate the Web site with a component.

Task ***To create a Web site on the target system at run time:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. Right-click the **Web Sites** explorer and click **Add Web Site**. InstallShield adds a new Web site.
3. Select the Web site to configure its settings.

Tip • *InstallShield lets you specify the TCP port and site numbers for a Web site in your project. These settings help determine whether a new Web site is created or an existing one is updated at run time. To learn more, see* *Configuring the TCP Port and Site Numbers*.

Task ***To create an application on the target system at run time:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, right-click the Web site that should contain the application and click **New Application**. InstallShield adds a new application.
3. Select the application to configure its settings.

Task ***To create a virtual directory on the target system at run time:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, right-click the Web site that should contain the virtual directory and click **New Virtual Directory**. InstallShield adds a new virtual directory.

3. Select the virtual directory to configure its settings.

Note • *To learn how Web sites, applications, and virtual directories are associated with components and features, see Feature and Component Associations for IIS Support.*

Scanning an IIS Web Site and Importing Its Settings into an InstallShield Project

Edition • *The ability to import IIS data into an InstallShield project is available only in the Premier edition of InstallShield.*

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

InstallShield includes support for scanning an existing IIS Web site, recording IIS data about the Web site, and importing those settings into the Internet Information Services view of an InstallShield project. Once you have imported the data into a project, you can make changes to the IIS settings as needed, and build an installation that creates or maintains an IIS Web site.

The IIS scanner (**IISscan.exe**) is a command-line tool that scans an IIS Web site and records the values of the settings that you can configure in the Internet Information Services view in InstallShield. **IISscan.exe** creates an XML file that contains all of the values. You can use this XML file to import the values into the Internet Information Services view.

IISscan.exe is installed in the following location:

*`InstallShield Program Files Folder`***\System\iisscan.exe**

Task ***To scan an existing Web site and record all of the IIS data that can be configured in the Internet Information Services view:***

1. Place the **IISscan.exe** tool on the server that has the IIS Web site that you want to scan.
2. At the command prompt, enter the command-line statement that launches the IIS scanner to scan a specific Web site. Following is a sample statement:

   ```
   iisscan.exe -website "Site Name" -outfile "C:\PathToFile\FileName.xml"
   ```

   For more information about **IISscan.exe**, see IISscan.exe.

The IIS scanner creates an XML file that contains the IIS data for the specified Web site.

Task ***To import the data from the XML file into an InstallShield project:***

1. Obtain the XML file that the IIS scanner created, and place it in a location that can be accessed from the machine that has InstallShield.
2. In the View List under **Server Configuration**, click **Internet Information Services**.
3. Right-click the **Web Sites** explorer and click **Import Web Site**. The **Open** dialog box opens.
4. Select the XML file that the IIS scanner created, and then click **Open**.

InstallShield imports the Web site and its settings into the Internet Information Services view. All of the Web site's settings, virtual directories, and applications are configured in this view during the import.

Tip *• InstallShield lets you configure filters if you want to prevent certain IIS data—such as Web sites, applications, virtual directories, application pool, or any of their settings—from ever being imported into the Internet Information Services view. To learn more, see* *Filtering IIS Data When Importing a Web Site and Its Settings into an InstallShield Project**.*

Filtering IIS Data When Importing a Web Site and Its Settings into an InstallShield Project

Edition *• The ability to import IIS data into an InstallShield project is available only in the Premier edition of InstallShield.*

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

When you use an XML file that is generated by the IIS scanner to import an IIS Web site and its settings into the Information Services view, you may find that certain IIS settings are imported, even though you do not need them to be configured in your installation. To avoid having these settings configured every time that you use an XML file to import IIS settings, you can edit **Filters.xml**. This file enables you to specify any IIS settings that you want the import to ignore.

Filters.xml is in the following location:

*`InstallShield Program Files Folder`***\Support**

The file must remain in this location after you edit it; otherwise, the IIS import will not work properly.

Tip *• You can also use the* **Filters.xml** *file to control which registry items should be excluded during COM extraction. For more information, see* *Filtering Registry Changes for COM Extraction**.*

The **Filters.xml** *file also lets you control which files should be included or excluded during dependency scanning. For more information, see Filtering Files in Dependency Scanners.*

Excluding IIS Data

The <Exclude> element in the **Filters.xml** file is where you add subelements for each of the IIS items (Web sites, applications, virtual directories, and application pools) that you do not want to be imported. You can also add subelements for each of the IIS settings that you do not want to be imported. Any items and settings that are listed here will not be added to the Internet Information Services view of your project.

Examples for Specifying IIS Data in the <Exclude> Element

To prevent an application pool named TestAppPool from being imported into the Internet Information Services view, add the following line within the <Exclude> element.

```
<IisScanItem name="TestAppPool" type="4"/>
```

The type is required. The following table shows the available IIS item types:

Table 4-3 • Recognized Attributes for the type Attribute of the <IisScanItem> Subelement

Attribute Value	Description
1	Web site
2	Application
3	Virtual directory
4	Application pool

To prevent an IIS setting with a specific value from being imported into the Internet Information Services view, add the following line within the <Exclude> element.

```
<IisScanProperty name="PropertyName" value="PropertyValue"/>
```

If this line is within the <Exclude> element, InstallShield blocks a setting named **PropertyName** with a value that equals or contains **PropertyValue**.

Note • *The property names and values are not Windows Installer properties and values. They are abbreviations for the names and values of settings that InstallShield lets you configure in the Internet Information Services view.*

Important • *Ensure that your XML code is well formed; if its not well formed, all of the filters fail. In most cases, you can identify improperly formed XML code by opening the* **Filters.xml** *file in Internet Explorer. You should be able to expand and contract the <Filters>, <Include>, and <Exclude> elements; if you cannot, check the code for errors.*

If you add subelements to the <Exclude> or <Include> elements, be sure that you do not place them within the commented-out section, since InstallShield ignores that area of the **Filters.xml** *file.*

The following sample XML code shows the format of the **Filters.xml** file:

```
<Filters>
 <Include>
  <!--Instructions on how to add files to this element.
  -->
  <File name="mfc42.dll" We="needthis"/>
 </Include>
 <Exclude>
  <!--Instructions on how to add files to this element.
  -->
  <Registry key="HKEY_CLASSES_ROOT\Interface\{00020404-0000-0000-C000-000000000046}"/>
  <File name="12520437.cpx" path="[SystemFolder]" wrp="4.0-10.0" />
  <File name="12520850.cpx" path="[SystemFolder]" wrp="4.0-10.0" />
  <IisScanProperty name="AppMappings" value="*;;*;0" />
  <IisScanProperty name="CustomErrors" value="C:\WINDOWS\help\iisHelp\common" />
 </Exclude>
</Filters>
```

Creating a Nested Virtual Directory



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

You can create a virtual subdirectory under an existing virtual directory.

You can also create a virtual subdirectory under a virtual directory that is being installed as part of your installation. The parent virtual directory must be installed before the virtual subdirectory.



Project *• For InstallScript projects, if a parent virtual directory should be installed before its child virtual subdirectory, the component that contains the parent virtual directory must be installed before the component that contains the child virtual subdirectory.*



Task ***To create a virtual directory under an existing virtual directory:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, select the Web site that should contain the nested virtual directory.
3. Right-click the new Web site and click **New Virtual Directory**. InstallShield adds a new virtual directory.
4. Click the **General** tab.
5. In the **Name** setting, indicate the name of the existing directory, as well as the name of the nested virtual subdirectory that you want to create. Separate both names with a slash.

   For example, to create a virtual directory called **MySubDirectory** under the existing virtual directory called **VirtualDirectory**, enter the following:

VirtualDirectory/MySubDirectory

Note • *If the parent directory does not already exist on the target system, the target system displays an error when the end user opens the directory in the IIS Manager.*

Configuring the TCP Port and Site Numbers

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

InstallShield lets you specify the TCP port and site numbers for a Web site in your project. These settings help determine whether a new Web site is created or an existing one is updated at run time. They also affect whether the Web site settings that are configured in the Internet Information Services view are applied to a Web site on the target system.

Task ***To specify the TCP port and site numbers for a Web site:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, select the Web site that you want to configure.
3. Click the **Web Site** tab.
4. In the **TCP Port Number** and **Site Number** boxes, enter the appropriate numbers.

Run-Time Behavior

At run time, the installation creates the Web site, applications, and virtual directories according to the following rules:

Table 4-4 • Run-Time Results for Various Sample TCP Port Number and Site Number Setting Values

TCP Port Number Setting in InstallShield	Site Number Setting in InstallShield	Result at Run Time
0	Any non-zero number	Since the TCP Port Number setting is 0 but the Site Number setting is not 0, the installation installs the Web site's applications and virtual directories to the first site number on the system. The specified site number is ignored. For example, if the first site number on the target system is 1, the installation installs the Web site's applications and virtual directories under site number 1, even if a different non-zero number such as 3 is specified for the Site Number setting. The installation does not apply any of the Web site settings that are configured in the Internet Information Services view to the Web site on the target system.
80 (a non-zero number)	0 (the default value)	If the specified TCP port exists on the target system, the installation installs the applications and virtual directories to the Web site that is running on the TCP port—in this example, port 80. The installation does not apply any of the Web site settings that are configured in the Internet Information Services view to the Web site on the target system. If the TCP port does not exist on the target system, a new Web site is created with the Web site settings that are configured in the project. In addition, the installation installs the Web site's applications and virtual directories.

Table 4-4 • Run-Time Results for Various Sample TCP Port Number and Site Number Setting Values (cont.)

TCP Port Number Setting in InstallShield	Site Number Setting in InstallShield	Result at Run Time
81 (a non-zero number)	3 (a non-zero number)	If the specified TCP port and site number exist on the target system, the installation installs the applications and virtual directories under the Web site whose site number matches the specified one—in this example, site number 3. The installation does not apply any of the Web site settings that are configured in the Internet Information Services view to the Web site on the target system. If the TCP port exists on the target system but the site number does not, the installation installs the new Web site, plus its applications and virtual directories, to the existing port with the new site number. In addition, the installation configures the Web site's properties that are set in the Internet Information Services view. If the TCP port does not exist on the target system but the site number does, the installation installs the new Web site, plus its applications and virtual directories, to this TCP port. In addition, the installation configures the Web site's properties that are set in the Internet Information Services view.

Installing an IIS Web Site and Its Applications and Virtual Directories to the Next Available New Site Number

The following procedures demonstrate how to install an IIS Web site and its applications and virtual directories to the next available new site number. Note that the procedure differs, depending on the project type.



Task ***To install to the next available new site number for a Basic MSI, DIM, InstallScript MSI, or Merge Module project:***

1. Create a new Windows Installer property and set its value to **INSTALLSHIELD_IIS_NEXT_NEW_SITE_NUMBER**:
   a. In the View List under **Behavior and Logic**, click **Property Manager**.
   b. Click the **New Property** button. InstallShield adds a new row at the bottom of the view.
   c. In the **Name** column, type a new property; for example:

      MYPROPERTY

   d. In the **Value** column, enter the following value:

      INSTALLSHIELD_IIS_NEXT_NEW_SITE_NUMBER
2. Specify the property for the Web site's site number:
   a. In the View List under **Server Configuration**, click **Internet Information Services**.

b. In the **Web Sites** explorer, select the Web site that you want to configure.

c. For the **Site Number** setting, enter the property that you created in step 1c. It must be in uppercase letters and enclosed within brackets; for example:

[MYPROPERTY]

At run time, the installation installs the Web site and its applications and virtual directories to the next available new site number.

Task ***To install to the next available new site number during an InstallScript installation:***

1. Create a new string entry and set its value to **INSTALLSHIELD_IIS_NEXT_NEW_SITE_NUMBER**:
   a. In the View List under **User Interface**, click **String Editor**.
   b. Click the **New String Entry** button. The **String Entry** dialog box opens.
   c. In the **String Identifier** box, enter a new variable name; the variable name must be in uppercase letters; for example:

   MYPROPERTY

   d. In the **Value** box, enter the following value:

   INSTALLSHIELD_IIS_NEXT_NEW_SITE_NUMBER
2. Specify the variable for the Web site's site number:
   a. In the View List under **Server Configuration**, click **Internet Information Services**.
   b. In the **Web Sites** explorer, select the Web site that you want to configure.
   c. For the **Site Number** setting, enter the variable that you created in step 1c. It must be in uppercase letters.

At run time, the installation installs the Web site and its applications and virtual directories to the next available new site number.

Specifying the IIS Host Header Name for a Web Site

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

InstallShield lets you specify the host header name to identify an IIS Web site that is installed during your installation. Host headers (also known as domain names) enable you to assign more than one Web site to an IP address on a Web server.

Task ***To specify a host header name for a Web site:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, select the Web site whose host header name you want to specify.
3. For the **Host Header Name** setting, type the host header name that you want to use. For example:

   www.mycompany.com

Specifying the SSL Certificate for a Web Site

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

A server certificate enables users to authenticate your Web server, check the validity of the Web content, and establish a secure connection. InstallShield lets you include a server certificate for a Web site in your installation so that it can be installed at run time.

Task ***To specify a SSL certificate that should be installed for a Web site:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, select the Web site whose SSL certificate you want to specify.
3. For the **SSL Certificate** setting, click the ellipsis button (**...**). The **Open** dialog box opens.
4. Select the security certificate file (.cer or .pfx) that you want to be installed, and then click **Open**.
5. If the certificate requires a password, specify it for the **SSL Certificate Password** setting.

InstallShield stores the .cer file in the **Binary** table. At run time, when the installation installs the Web site and its virtual directories, it also installs the SSL certificate.

Adding Files to an IIS Virtual Directory

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*
- *Merge Module*

Task ***To add a file to an IIS virtual directory:***

1. Add an IIS Web site to your project if you have not already done so. InstallShield automatically adds the predefined path **[IISROOTFOLDER]** to the **Files and Folders** view.
2. In the View List under **Application Data**, click **Files and Folders**.
3. In the **Feature** list, select the feature with which you want the file associated.
4. In the **Destination computer's folders** pane, add the file to the **[IISROOTFOLDER]** folder, or a subfolder of the **[IISROOTFOLDER]** folder.
5. In the View List under **Server Configuration**, click **Internet Information Services**.
6. Create a new virtual directory.
7. In the **Web Sites** explorer, click the virtual directory that you created.
8. In the **Content Source Path (Local or UNC)** setting, click the ellipsis button (**...**). The **Browse for Directory** dialog box opens. In a Basic MSI or InstallScript MSI project, this dialog box enables you to select a Windows Installer property (such as [IISROOTFOLDER]) or create a new one. In an InstallScript project, this dialog box enables you to select an InstallScript variable (such as <IISROOTFOLDER>) or create a new one. By default, the files are stored in IISROOTFOLDER.
9. Enter the same target directory as the one that contains the new file that you added in the **Files and Folders** view.
10. Click **OK**.

At installation, files are copied to the target directory folder. In addition, the virtual directory is configured for that folder on the target system if IIS is present.

Removing Applications and Virtual Directories from the Internet Information Services View

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Task ***To remove an application or a virtual directory from your installation:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, right-click the application or virtual directory and click **Delete**.

Adding Application Pools

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

An application pool is a group of configuration settings that specify how one or more Web applications are routed to one or more worker processes. InstallShield lets you create application pools and associate Web sites, applications, and virtual directories with those application pools.

Project *• If you are creating an application pool in an InstallScript project and you want the application pool to be associated with a virtual directory, application, or Web site, you must create the application pool and the virtual directory, application, or Web site within the same component. If you do this, the application pool is created before the virtual directory, application, or Web site is created. This is a requirement of IIS 6 and later, and it is not a limitation for Basic MSI, DIM, InstallScript MSI, or Merge Module projects.*

Note *• Application pools are available only on machines with IIS 6.0 or later installed.*

Task ***To add a new application pool in the Internet Information Services view:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. Right-click the **Application Pools** explorer and click **Add Application Pool**. InstallShield adds a new application pool item.
3. Rename the new application pool item, and configure its settings.

Removing Application Pools from the Internet Information Services View

Project *• This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*



***Project** • To remove an application pool from your installation:*

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Application Pools** explorer, right-click the application pool and click **Delete**.

Adding Web Service Extensions



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

A Web service extension is an IIS feature that extends the basic IIS functionality beyond serving static content. Examples of Web service extensions are active server pages (.asp), ASP.NET, and server-side includes (SSI).

InstallShield lets you add Web service extensions to your installation.



***Note** • Web service extensions are available only on machines with IIS 6.0 or later installed.*

On systems with IIS 7, Web service extensions require that the IIS Metabase and IIS 6 Configuration Compatibility feature be installed. For more information, see Version-Specific Information for IIS Support in InstallShield.



Task ***To add a web service extension to your installation:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. Right-click the **Web Service Extensions** explorer and click **Add Web Service Extension**. InstallShield adds a new Web service extension.
3. Rename the new Web service extension item, and configure its settings.

Removing Web Service Extensions from the Internet Information Services View

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Task ***To remove a Web service extension from your installation:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Service Extension** explorer, right-click the Web service extension and click **Delete**.

Feature and Component Associations for IIS Support

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

There is a one-to-many relationship between components and IIS data (that is, Web sites, applications, virtual directories, application pools, or Web service extensions). Therefore, you can associate more than one IIS element with a single component. If a component is being installed, any Web sites, applications, virtual directories, application pools, and Web service extensions that are associated with the component are created at run time.

Project *• If a Web site is associated with a component in an InstallScript project, any applications or virtual directories that are added to that Web site must be associated with the same component. Therefore, if you try to change the component for a Web site that contains one or more applications or virtual directories, InstallShield displays a message box to inform you that it will also make the same component change for all of that Web site's applications and virtual directories; InstallShield also displays this message box if you try to change the component for any applications or virtual directories in a Web site. In either case, the message box enables you to proceed or cancel the component change.*

Note that for Basic MSI, DIM, InstallScript MSI, and Merge Module projects, keeping a Web site in the same component as all of the Web site's applications and virtual directories is not required, but it is recommended.

If you are creating an application pool in an InstallScript project and you want the application pool to be associated with a virtual directory, application, or Web site, you must create the application pool and the virtual directory,

application, or Web site within the same component. If you do this, the application pool is created before the virtual directory, application, or Web site is created. This is a requirement of IIS 6 and later, and it is not a limitation for Basic MSI, DIM, InstallScript MSI, or Merge Module projects.

At any time, you can use the Internet Information Services view to change which component contains the Web site, application, virtual directory, application pool, or Web service extension.

Task ***To associate a Web site, application, virtual directory, application pool, or Web service extension with a different component in your project:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the explorer, click the Web site, application, virtual directory, application pool, or Web service extension that you want to associate with a component.
3. In the **Component** setting, select the name of the existing component that should contain the selected IIS data. You can click the ellipsis button (...) to select an existing component or create a new one for the IIS data.

Tip *• If you delete a component in your project, any Web sites, applications, virtual directories, application pools, and Web service extensions that are associated with the component are simultaneously removed from your project.*

Uninstalling Web Sites, Applications, and Virtual Directories

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Web sites that are created by an installation are never removed unless both of the following conditions are true:

- The Web site no longer contains applications or virtual directories.
- The value for the **Delete on Uninstall** setting for the Web site in the Internet Information Services view is Yes.

Note *• If a Web site is associated with a component, the* ***Delete on Uninstall*** *setting for the Web site corresponds with the Permanent setting for that component in a Basic MSI, DIM, InstallScript MSI, or Merge Module project, or with the Uninstall setting for that component in an InstallScript project. That is, if you select or clear the* ***Delete on Uninstall*** *setting for a Web site, InstallShield automatically updates the value of the component's Permanent setting or Uninstall setting, as appropriate.*

If a component is uninstalled, all of its Web sites, applications, and virtual directories are uninstalled. A component is not uninstalled if it is configured to be permanent. For more information, see Specifying Whether a Component's Files and Other Associated Data Are Uninstalled During Uninstallation.

Uninstalling Web Service Extensions and Application Pools



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*



Note *• Web service extensions, application pools, and associated properties are available only on machines with IIS 6.0 or later installed.*

On systems with IIS 7, Web service extensions require that the IIS Metabase and IIS 6 Configuration Compatibility feature be installed. For more information, see Version-Specific Information for IIS Support in InstallShield.

Any Web service extensions and application pools installed in the IIS Manager that have identical names as the Web service extensions and application pools in an installation will always be uninstalled when the features with which they are associated are uninstalled unless their components are marked as permanent. Even if the Web service extension or application pool was originally created by something other than the installation, it will be removed from the target system upon uninstallation if the names (of the Web service extensions and application pools) match those in the installation. One exception is that the default application pool, named DefaultAppPool, will never be removed by the uninstallation.



Task ***To configure a Web service extension to be uninstalled during uninstallation:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Service Extensions** explorer, select the Web service extension.
3. For the **Mark Component as Permanent** setting, select **No**.



Task ***To configure an application pool to be uninstalled during uninstallation:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Application Pool** explorer, select the application pool.
3. For the **Mark Component as Permanent** setting, select **No**.

If Yes is selected for the Mark Component as Permanent setting, the Web service extension or application pool is left on the target system after uninstallation.



Tip *• Changing the value of the Mark Component as Permanent setting affects the value of the Permanent setting (in Basic MSI, DIM, InstallScript MSI, and Merge Module projects) or the Uninstall setting (in InstallScript projects) for the component that contains the Web service extension or the application pool. Therefore, if you select Yes for the Mark*

Component as Permanent and the component contains other data, such as files, shortcuts, and registry entries, the IIS data, as well as the other data in the component, are not uninstalled during uninstallation.

Setting the ASP.NET Version for a Web Site or Application



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

InstallShield lets you set the ASP.NET version for a Web site or application in your installation. If you specify the ASP.NET version, after the Web site or application is created, the installation runs the ASP.NET IIS Registration tool (**Aspnet_regiis.exe**) to map the Web site or application to the version that you specify.

If you specify the ASP.NET version for a Web site, IIS uses that value for the Web site that is created at run time, as well as for any of its applications.



Important • *Microsoft does not recommend using the* **Aspnet_regiis.exe** *tool on Windows Vista and Windows Server 2008 systems because it has limited capabilities. As a result, you may need to manually define application mappings in the Internet Information Services view. To learn more, see Defining Application Mappings for a Web Site, Application, or Virtual Directory.*

ASP.NET 3.0 does not include the **Aspnet_regiis.exe** *tool. Therefore, you cannot set the ASP.NET version to version 3 of ASP.NET.*



Task

To specify the ASP.NET version for a Web site or application in your project:

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, select the Web site or application whose ASP.NET version you want to specify. The settings for the Web site or application are displayed on the right.
3. For the **ASP.NET Version** setting, enter the complete version number of the .NET Framework that is required by your application, or select it from the list.

   For example, to specify version 2 of ASP.NET, type **2.0.50727**. To specify version 1.1 of ASP.NET, type **1.1.4322**.



Tip • *If the installation may be run on a 64-bit version of Windows with the .NET Framework, you can use the ASP.NET Platform setting for a Web site or application to specify which ASP.NET platform (32 bit or 64 bit) should be used to map the Web site or application to the ASP.NET version. Note that it is not possible to register 32-bit ASP.NET on a 64-bit system unless the Enable32BitAppOnWin64 property is set to true on the system.*

Considerations for Supporting IIS 6 on 64-Bit Platforms



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

If your IIS installation may be run on a 64-bit version of Windows with the .NET Framework, you can use the ASP.NET Platform setting in the Internet Information Services view for a Web site or application. This setting lets you specify which ASP.NET platform should be used to map the Web site or application to the ASP.NET version. By default, this setting is not configured in InstallShield; if you do not change the value of this setting, the installation assumes that 32-bit support is needed.

Depending on the value that you configure for the ASP.NET Platform and ASP.NET Version settings, the following behavior occurs by default at run time:

- If the target system has a 32-bit version of Windows, the 32-bit version of the specified ASP.NET version is used.
- If the target system has a 64-bit version of Windows, and 64 bit was specified for the ASP.NET platform, the Enable32bitAppOnWin64 property on the target system needs to be set to False. Otherwise, using the 64-bit version of ASP.NET on a system that is configured to support the 32-bit version would cause ASP.NET to become corrupted. Thus, the installation is aborted to prevent damage to ASP.NET services.
- If the target system has a 64-bit version of Windows, and 32 bit was specified for the ASP.NET platform, the Enable32bitAppOnWin64 property on the target system needs to be set to True. Otherwise, using the 32-bit version of ASP.NET on a system that is configured to support the 64-bit version would cause ASP.NET to become corrupted. Thus, the installation is aborted to prevent damage to ASP.NET services.

You can set the Enable32bitAppOnWin64 property on a target system that has IIS 7 by configuring the Enable 32-Bit Applications setting for an application pool in the Internet Information Services view. At run time, the installation sets the Enable32bitAppOnWin64 property as appropriate for the application pool that is being installed.

On systems with IIS 6, the Enable32bitAppOnWin64 property is a global property that affects all Web sites and applications on an IIS server. Since changing the value of that property at installation run time might cause other already installed Web sites and applications on that server to fail, installations that are created with InstallShield do not modify the Enable32bitAppOnWin64 property at run time if the target system has IIS 6.

At run time, you can have your installation check the Enable32bitAppOnWin64 property on systems that have IIS 6. Depending on the requirements for your product and the results of that check, you may want the installation to skip a particular component that contains a 32-bit IIS 6 application, or a 64-bit IIS application, for example, and proceed with the rest of the file transfer.

InstallShield includes a sample Windows Installer DLL file that detects how the Enable32bitAppOnWin64 property is set. Release and debug versions of the DLL file, as well as the Visual Studio 2008 C++ project that was used to create the file, are installed to the following location:

*`InstallShield Program Files Folder`***\Samples\WindowsInstaller\Detect IIS6 Compatibility**

To use the Windows Installer DLL file in your project, you first need to create a custom action that calls the **DetectIIS6AppPool32BitSupport** function in the **DetectIIS6Compat.dll** file.

Task ***To add a custom action that checks for support for 32-bit applications on systems that have IIS 6:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI or InstallScript MSI projects) or **Custom Actions** (in DIM or Merge Module projects).
2. In the center pane, right-click the **Custom Actions** explorer, point to **New MSI DLL**, and click **Stored in Binary table**. InstallShield adds a new custom action.
3. Change the name of the action to something like **DetectIIS6AppPool32BitSupport**.
4. Configure the custom action's settings in the right pane:
   - In the **DLL Filename** setting, specify the path to the **DetectIIS6Compat.dll** file. The typical path is:

     <ISProductFolder>\Samples\WindowsInstaller\Detect IIS6 Compatibility\DetectIIS6Compat.dll
   - In the **Function Name** setting, enter the following name:

     DetectIIS6AppPool32BitSupport
   - In the **In-Script Execution** setting, select **Immediate Execution**.
   - Schedule the custom action as needed by selecting the appropriate option in one or more sequence settings.
   - In the condition setting, enter the following condition:

     VersionNT<600

     This condition ensures that the custom action is not run on Windows Server 2008 or later or on Windows Vista or later, since these operating systems would have IIS 7 or later.

At run time, the custom action sets the Windows Installer property **ISIIS6APPPOOLSUPPORTS32BIT** under certain conditions:

Table 4-5 • Windows Installer Property for Checking for 32-Bit Application Support on IIS 6 Systems

Target System Environment	Result
32-bit Windows, IIS 6 (The value of Enable32bitAppOnWin64 is not relevant.)	**ISIIS6APPPOOLSUPPORTS32BIT** is set to 1.
64-bit Windows, IIS 6, Enable32bitAppOnWin64=true	**ISIIS6APPPOOLSUPPORTS32BIT** is set to 1.
64-bit Windows, IIS 6, Enable32bitAppOnWin64=false	**ISIIS6APPPOOLSUPPORTS32BIT** is not set.

You can use the **ISIIS6APPPOOLSUPPORTS32BIT** property in conditional statements. For example, if your product cannot be used on an IIS 6 server without 64-bit application support, enter the following conditional statement in the Install Condition setting of the General Information view:

Not ISIIS6APPPOOLSUPPORTS32BIT

As an alternative, you can create an error custom action. At run time, an error would be displayed if the condition for that custom action is true. Thus, if your product cannot be used on an IIS 6 server without 64-bit application support, use the following condition for an error custom action:

ISIIS6APPPOOLSUPPORTS32BIT=1

If you have a component that should be installed only on IIS 6 servers that have 32-bit application support enabled, enter the following conditional statement in the Condition setting of the Components view:

ISIIS6APPPOOLSUPPORTS32BIT=1

Defining Application Mappings for a Web Site, Application, or Virtual Directory



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

InstallShield lets you define mappings between file name extensions and the applications that process those files.



Task ***To add, edit, or remove an application mapping:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, select the Web site, application, or virtual directory that you want to configure.
3. In the **Application Mappings** setting, click the ellipsis button (**...**). The **Application Mappings** dialog box opens.
4. Do one of the following:
   - To add a new mapping, click the **Add** button. The **Application Extension Mapping** dialog box opens. For more information, see Application Extension Mapping Dialog Box.
   - To change an existing mapping, select the one that you want to edit, and then click the **Edit** button.
   - To delete an existing mapping, select it and then click the **Delete** button.

Specifying Timeout Parameters for a Web Site, Application, or Virtual Directory



Project *• This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

InstallShield lets you specify timeout parameters for a Web site, application, or virtual directory.

Task ***To specify timeout parameters:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, select the Web site, application, or virtual directory that you want to configure.
3. In the **Application settings** area, click the **Configuration** button. The **Application Mappings** dialog box opens.
4. In the **Session Timeout (minutes)** and the **ASP Script Timeout (seconds)** settings, specify the appropriate timeout values.

Determining If a Target System Has IIS 6 or Earlier or the IIS 6 Metabase Compatibility Feature

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

At run time, you can have your installation detect if the IIS Metabase and IIS 6 Configuration Compatibility feature is installed on a target system, or if IIS 6 or earlier is installed. Depending on the requirements for your product and the results of that detection, you may want the installation to exit and display an error message.

For example, Web service extensions can be installed on systems that have IIS 6. On systems that have IIS 7, Web service extensions can be installed only if the IIS Metabase and IIS 6 Configuration Compatibility feature is installed. Thus, you might want to configure your installation to verify that IIS 6 is present or the IIS 6 compatibility feature is installed; if either of those conditions are false, the installation would exit and display an error message.

InstallShield includes a sample Windows Installer DLL file that detects if either of the following are true:

- The target system has IIS 7 and the IIS Metabase and IIS 6 Configuration Compatibility feature is installed.
- The target system has IIS 6 or earlier.

Release and debug versions of the DLL file, as well as the Visual Studio 2008 C++ project that was used to create the file, are installed to the following location:

*`InstallShield Program Files Folder`***\Samples\WindowsInstaller\Detect IIS6 Compatibility**

To use the Windows Installer DLL file in your project, you first need to create a custom action that calls the **DetectIIS6Interfaces** function in the **DetectIIS6Compat.dll** file.



Task ***To add a custom action that checks for the IIS Metabase and IIS 6 Configuration Compatibility feature and for IIS 6 and earlier:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI or InstallScript MSI projects) or **Custom Actions** (in DIM or Merge Module projects).
2. In the center pane, right-click the **Custom Actions** explorer, point to **New MSI DLL**, and click **Stored in Binary table**. InstallShield adds a new custom action.
3. Change the name of the action to something like **DetectIIS6Interfaces**.
4. Configure the custom action's settings in the right pane:
   - In the **DLL Filename** setting, specify the path to the **DetectIIS6Compat.dll** file. The typical path is:

     <ISProductFolder>\Samples\WindowsInstaller\Detect IIS6 Compatibility\DetectIIS6Compat.dll
   - In the **Function Name** setting, enter the following name:

     DetectIIS6Interfaces
   - In the **In-Script Execution** setting, select **Immediate Execution**.
   - Schedule the custom action as needed by selecting the appropriate option in one or more sequence settings.

If the IIS Metabase and IIS 6 Configuration Compatibility feature is installed or the target system has IIS 6 or earlier, the Windows Installer property **ISIISMETABASECOMPATPRESENT** is set to a value of 1. If the target system has IIS 7 or later and the compatibility feature is not installed, the property is not set.

You can use the **ISIISMETABASECOMPATPRESENT** property in conditional statements. For example, if your product cannot be used on a system without the IIS Metabase and IIS 6 Configuration Compatibility feature or IIS 6 or earlier, enter the following conditional statement in the Install Condition setting of the General Information view:

ISIISMETABASECOMPATPRESENT

As an alternative, you can create an error custom action. At run time, an error would be displayed if the condition for that custom action is true. Thus, if your product cannot be used on a system without the IIS Metabase and IIS 6 Configuration Compatibility feature or IIS 6 or earlier, use the following condition for an error custom action:

Not ISIISMETABASECOMPATPRESENT

If you have a component that should be installed only on systems that have the IIS Metabase and IIS 6 Configuration Compatibility feature or IIS 6 or earlier, enter the following conditional statement in the Condition setting of the Components view:

ISIISMETABASECOMPATPRESENT=1

Configuring Advanced IIS Settings



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Note • *The advanced IIS settings apply to IIS 6 and earlier. IIS 7 ignores these settings.*

When you select a Web site, application, or a virtual directory in the Web Sites explorer in the Internet Information Services view, the right pane exposes the most common IIS settings. You can use the Advanced area at the bottom of the right pane to configure other IIS settings that are not exposed in the other areas.

Task ***To configure a setting that is not exposed in the Internet Information Services view:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, select the Web site, application, or virtual directory whose advanced settings you want to configure.
3. In the **Other IIS Properties** setting, click the ellipsis button (**...**). The **Other IIS Properties** dialog box opens.
4. In the property list, select the property whose value you want to change, and then click the **Change Value** button. The **Edit IIS Metabase Value** dialog box opens.
5. In the **Value** box, specify the new value.

If you change the default value for any advanced property, InstallShield adds the property, the value that you specify, and the additional required information to the **ISIISProperty** table. For more information, see ISIISProperty Table.

Configuring Custom Error Messages for a Web Site, Application, or Virtual Directory

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

When an end user tries to connect to a Web site and a hypertext transfer protocol (HTTP) error occurs, the end user's browser displays a default message describing the error. You can have your installation configure IIS so that it uses custom error messages instead of the default error messages by mapping the HTTP error codes to a file or a URL.

Task ***To configure custom error messages for a Web site, application, or virtual directory:***

1. Create a file that contains your custom error message and add it to your installation.
2. In the View List under **Server Configuration**, click **Internet Information Services**.

3. Select the Web site, application, or virtual directory for which you want to customize HTTP error messages. The settings for the Web site, application, or virtual directory are displayed on the right.
4. In the **Custom Errors** setting, click the ellipsis button (**...**). The **Custom Errors** dialog box opens.
5. Select the HTTP error code that you want to change and then click the **Edit Properties** button. The **Error Mapping Properties** dialog box opens.
6. To map the error code to a file:
   a. In the **Message Type** list, select **File**.
   b. In the **File** box, type the path and file name that points to the custom error message in your installation, or click the browse button to locate the file.

   To map the error code to a URL:
   a. In the **Message Type** list, select URL.
   b. In the **URL** box, type the URL that points to your custom error message.

Using Windows Installer Properties to Dynamically Modify IIS Settings



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For Basic MSI and InstallScript MSI projects, you can configure an IIS setting dynamically at run time through the use of Windows Installer properties. This enables you to let end users specify the name of the virtual directory, the TCP port, the site number, or other IIS settings for the Web sites, applications, virtual directories, application pools, and Web service extensions that they are installing on the target machine.

Windows Installer uses **MsiFormatRecord** to resolve the property at run time. The installation writes the property and its value to the following registry key so that the value is available during uninstallation and repair:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\InstallShield Uninstall Information*{ProductCode}*

Therefore, if your Web site is installed to an end user–specified site number, the Web site, its applications, and its virtual directories can be successfully uninstalled from that site number.



Tip • *If you do not want the IIS data to be stored in the registry, set the **IS_IIS_DO_NOT_USE_REG** property in your project.*

If you use a property for any of the passwords in the Internet Information Services view, the passwords are encrypted when they are stored in the registry.

Example

The following procedure demonstrates how to let end users specify during installation the user name used for anonymous access to the Web site. Note that you can substitute a hard-coded value with a property for any of the IIS settings in the Internet Information Services view that allow you to enter characters. The property that you specify in this view must be enclosed within square brackets, and the property name must be all uppercase; for example, **[MYPROPERTY]**.

Step 5 of the procedure is slightly different, depending on the project type, since Windows Installer controls the user interface of Basic MSI installations, and the InstallScript engine controls the user interface of InstallScript MSI installations.



Task ***To let end users specify the user name:***

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, click the Web site whose user name end users should be able to specify for anonymous access.
3. For the **Enable anonymous access** setting, select **Yes**.
4. In the **Anonymous User Name** setting, type the following:

   [MYPROPERTY]
5. Use the property in a dialog. This part of the procedure depends on which project type you are using.
   - For Basic MSI projects:
     a. In the View List under **User Interface**, click **Dialogs**.
     b. In the **Dialogs** explorer, expand the **All Dialogs** folder, and click the language under the dialog that should contain the User Name control. As an alternative, you can add a new dialog.
     c. Add an **Edit Field** control to the dialog, and set its **Property** property to the following:

        MYPROPERTY
   - For InstallScript MSI projects:
     a. In the View List under **Behavior and Logic**, click **InstallScript**.
     b. Find the dialog code in the OnFirstUIBefore event for the dialog that should contain the User Name control, and add a call to the Windows Installer API function **MsiSetProperty**. For example, if you want the user name to be the name that the end user specifies on the CustomerInformation dialog, you would add an **MsiSetProperty** call as shown in the following lines of code:

        ```
        Dlg_SdCustomerInformation:
            nResult = SdCustomerInformation(szTitle, svName, svCompany, nUser);
            MsiSetProperty(ISMSI_HANDLE, "MYPROPERTY", svName);
            if (nResult = BACK) goto Dlg_SdWelcome;
        ```
6. Build your release.

Using InstallScript Text Substitution to Dynamically Modify IIS Settings

Project • *This information applies to InstallScript projects.*

For InstallScript projects, you can configure an IIS setting dynamically at run time through the use of text substitution string variables. This enables you to let end users specify the name of the virtual directory, the TCP port, the site number, or other IIS settings for the Web sites, applications, virtual directories, application pools, and Web service extensions that they are installing on the target machine. The InstallScript run-time code uses the **TextSubSubstitute** function to replace the string variable with the appropriate value.

Example

The following procedure demonstrates how to let end users specify during installation the user name used for anonymous access to the Web site. Note that you can substitute a hard-coded value with a text substitution string variable for any of the IIS settings in the Internet Information Services view that allow you to enter characters. The string variable that you specify in this view must be enclosed within angle brackets; for example, **<MYPROPERTY>**. The string variable that you specify is case-sensitive.

Task

To let end users specify the user name:

1. In the View List under **Server Configuration**, click **Internet Information Services**.
2. In the **Web Sites** explorer, click the Web site whose user name end users should be able to specify for anonymous access.
3. For the **Enable anonymous access** setting, select **Yes**.
4. In the **Anonymous User Name** setting, type the following:

   <MYPROPERTY>
5. In the View List under **Behavior and Logic**, click **InstallScript**.
6. Find the dialog code in the OnFirstUIBefore event for the dialog that should contain the User Name control, and add a call to the InstallScript function **TextSubSetValue**. For example, if you want the user name to be the name that the end user specifies on the CustomerInformation dialog, you would add a **TextSubSetValue** call as shown in the following lines of code:

```
Dlg_SdRegisterUser:
    szMsg = "";
    szTitle = "";
    //{{IS_SCRIPT_TAG(Dlg_SdRegisterUser)
    nResult = SdRegisterUser( szTitle, szMsg, szName, szCompany );
    TextSubSetValue("<MYPROPERTY>", szName, TRUE);
    //}}IS_SCRIPT_TAG(Dlg_SdRegisterUser)
    if (nResult = BACK) goto Dlg_SdLicense2;
```

7. Build your release.

Deploying Web Services on a Target Machine

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Deploying a Web service onto a target system requires copying the Web service–specific files to a particular location and assigning a virtual directory name for that folder so that the Web service can be accessed via HTTP.

Tip *• To learn how to add a virtual directory to your project, see Creating a Web Site and Adding an Application or a Virtual Directory.*

Task

To deploy a Web service on a target machine:

1. In the View List under **Application Data**, click **Files and Folders**.
2. Select the folder (target directory) for installing files on the target system. Add your files to this folder.
3. In the View List under **Server Configuration**, click **Internet Information Services**.
4. In the **Web Sites** explorer, select the virtual directory that is associated with the Web service.
5. In the **Content Source Path (Local or UNC)** setting, click the ellipsis button (**...**). The **Browse for Directory** dialog box opens. Enter the same target directory as the one that contains the new files that you added in the **Files and Folders** view.

At installation, files are copied to the target directory folder. In addition, the virtual directory is configured for that folder on the target system if IIS is present.

Adding IISROOTFOLDER Support

IISROOTFOLDER is an InstallShield directory variable that is used to determine the root directory of the Web server on a target system. If you are using IIS functionality in your installation project and you have added a Web site, then IISROOTFOLDER is automatically added.

Note *• All of the files added to the IISROOTFOLDER directory in the Files and Folders view are installed to the Web server's root directory on the target machine. If IIS is not installed, the files are copied to the root folder.*

IIS_WEBSITE_NAME Property

The **IIS_WEBSITE_NAME** property is obsolete. If this property exists from earlier project versions, the upgrader will handle it automatically. The upgrader will create a Web site and set the site number field to [IIS_WEBSITE_NAME]. For new Web sites, you can use any property or hard-coded number.

IIS_PORT_NUMBER Property

The **IIS_PORT_NUMBER** property is obsolete. If this property exists from earlier project versions, the upgrader will handle it automatically. The upgrader will create a Web site and set the port number field to [IIS_PORT_NUMBER]. When you create new Web sites, you can use any property or hard-coded number.

Preparing Installations for Maintenance and Uninstallation

InstallShield lets your users rerun your installation to change program features or reinstall or remove your application. When users select your application in Add or Remove Programs in the Control Panel, the installation displays dialog boxes that let users do any of the following:

- Install individual features that were not previously installed, and uninstall individual features.
- Reinstall your application with the settings selected during the first installation.
- Uninstall your application.

To modify, repair, or uninstall an application, the operating system must have some indication that the application is present. To allow this, an installation registers an application with the operating system so that it can be easily maintained or uninstalled. You enter the necessary information in the General Information view.

In all InstallShield installations, maintenance (that is, modification and repair) is handled automatically by default.

In a Windows Installer–based installation, uninstallation is handled automatically—with the sole exception of custom actions, which must either undo their own effect during uninstallation or have their effect undone by another custom action that runs only during uninstallation.

In an InstallScript or InstallScript MSI installation, many script functions' actions are logged for automatic uninstallation; your uninstallation script must handle those script function actions that are not logged.

Preparing an InstallScript Installation for Uninstallation

Setting up uninstallation functionality is simple, considering the complexity of the operations that result. **CreateInstallationInfo** (or **InstallationInfo**) and **MaintenanceStart** (or **DeinstallStart**) must be called. In an event-based script, **CreateInstallationInfo** and **MaintenanceStart** are called in the default **OnMoveData** event handler code.

There are other functions that play important roles in setting up uninstallation functionality. For example, all keys created using the **RegDBCreateKeyEx** function are recorded for uninstallation, unless logging is disabled. Also, if you call the **Disable** function to disable logging, remember to call **Enable** to enable the logging again for subsequent actions that must be recorded for uninstallation.

MaintenanceStart (or **DeinstallStart**) creates the necessary registry key and its necessary values. To create additional registry values under this key, call RegDBSetItem, or for custom values use code like the following:

```
RegDBSetDefaultRoot( HKEY_USER_SELECTABLE );
RegDBSetKeyValueEx( "Software\\Microsoft\\Windows\\Current Version\\Uninstall\\" + INSTANCE_GUID, "My
Custom Value", nType, szValue, nSize);
```



Note *• Uninstallation is available only if the initial installation calls* ***FeatureTransferData*** *or* ***FeatureMoveData****. (In an event-based script, the* ***FeatureTransferData*** *function is called automatically.) If your installation installs files by some other means (for example,* ***XCopyFile****) and you want to make uninstallation available to your end users, deselect all components and call* ***FeatureTransferData*** *or* ***FeatureMoveData****; while this call is executed, you can call* ***SdShowMsg*** *to display a message.*

Including Script Code to Be Executed During Uninstallation



***Note** • If you call **DeinstallStart** to enable uninstallation, the installation script is not executed during uninstallation. To execute script code during uninstallation, call **MaintenanceStart** instead. (In an event-based installation, **MaintenanceStart** is called in the default **OnMoveData** event handler code.)*

Event-Based Scripts

If your script is event-based, the following event handler functions are called during uninstallation:

- **OnBegin**
- **OnMaintUIBefore**
- **OnMoving**
- <Feature name>**Uninstalling** (for each installed feature)
- <Feature name>**Uninstalled** (for each installed feature)
- **OnMoved**
- **OnMaintUIAfter**
- **OnEnd**

The code in these event handler functions (whose default code you can override) is executed during the uninstallation. For instructions on executing certain event handler code only during uninstallation, or only during installation, see Executing Script Code Only During Uninstallation or Only During Installation.

Procedural Scripts

If your script is procedural (that is, uses a program block) and calls **MaintenanceStart** rather than **DeinstallStart** to enable uninstallation, the script is executed during uninstallation. For instructions on executing certain script code only during uninstallation, or only during installation, see Executing Script Code Only During Uninstallation or Only During Installation.

Executing Script Code Only During Uninstallation or Only During Installation

The installation script is executed during uninstallation and maintenance installations if **MaintenanceStart** rather than **DeinstallStart** is called to enable uninstallation. (In an event-based script, **MaintenanceStart** is called in the default **OnMoveData** event handler code.) To execute code only during uninstallation or maintenance installations, enclose it in the following if-then statement:

```
if MAINTENANCE then
    /* this code is executed only during
    uninstallation or maintenance installations */
endif;
```

To execute code only during a first installation, enclose it in the following if-then statement:

```
if !MAINTENANCE then
    /* this code is executed only during
```

```
      a first installation */
endif;
```

If code is not enclosed in either if-then statement, it is executed during uninstallation, maintenance installations, and first installations—with the following exceptions in event-based scripts:

- The code in the following event handlers is executed only during uninstallation and maintenance installations:

  OnMaintUIBefore

  OnMaintUIAfter

- The code in the following event handlers is executed only during the first installation:

  OnCCPSearch

  OnAppSearch

  OnFirstUIBefore

  OnFirstUIAfter

InstallScript Functions that Are Logged for Uninstallation



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI —if the InstallScript user interface (UI) style is the traditional style (which uses the InstallScript engine as an external UI handler)*

This information does not apply to InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

In general, if logging is enabled, all of the files, folders, registry entries, .ini file entries, shortcuts, shortcut folders, and services that are created or replaced while logging is enabled are logged for uninstallation. In addition, all of the InstallScript functions that make changes to the target system during InstallScript installations and InstallScript MSI installations are logged for uninstallation. Following are some common examples of functions that an installation logs for uninstallation by default:

- AddFolderIcon
- AddProfString
- CopyFile
- CreateInstallationInfo
- CreateProgramFolder
- FeatureMoveData
- FeatureTransferData
- RegDBSetAppInfo
- RegDBSetItem

- ReplaceProfString
- ServiceAddService
- ServiceStartService
- VerSearchAndUpdateFile
- VerUpdateFile
- WriteProfString
- XCopyFile

You can disable and re-enable logging by calling `Disable (LOGGING);` and `Enable (LOGGING);` in your InstallScript code. For more information, see Disable and Enable.

Maintenance/Uninstallation Feature

The maintenance/uninstallation feature installs the files needed for maintenance setups and uninstallation, including engine files; the files' target location is specified by the value of the system variable *DISK1TARGET*. This feature is automatically placed in your .cab files by the release builder and is not displayed in the InstallShield interface.

At run time the maintenance/uninstallation feature is automatically selected for all setup types. Calling **FeatureUpdate** deselects this feature. The maintenance/uninstallation feature can also be selected or deselected by calling **FeatureSelectItem** and passing the system variable *DISK1COMPONENT* as the function's second argument. For example, the following code deselects the feature:

```
FeatureSelectItem( MEDIA, DISK1COMPONENT, FALSE );
```

Removing Files that Were Created by Your Product

By default, files created by your product after the installation is complete are assumed to be user data, and are not removed when the user uninstalls your application.

Basic MSI Project

Records you create in the RemoveFile table (using Direct Editor) can specify additional files to remove. For example, if your application creates a file called Product.ini inside *INSTALLDIR* and you want the file removed when the component containing your main executable is removed, add a record with the following contents to the RemoveFile table:

Table 4-1 • Additions to Add to the RemoveFile Table

FileKey	remove_file
Component_	ProgramFiles
FileName	Product.ini
DirProperty	INSTALLDIR
InstallMode	2

InstallScript and InstallScript MSI Projects

In your script, you can call the DeleteFile and DeleteDir functions in uninstallation event handler functions to remove any files or directories you want removed.

Removing Registry Data Created by Your Product

By default, your product's uninstallation removes only data that was created by your installation program.

Basic MSI Project

The Registry explorer understands a special uninstallation flag that controls the registry data to be removed during uninstallation. In particular, using the "-" flag ("Uninstall entire key") on a registry key causes the key and all its values and subkeys to be removed during uninstallation.



Note *• The RemoveRegistry table—exposed in the Direct Editor—specifies data to be removed only when your product is installed. For details, see the Windows Installer Help Library.*

InstallScript MSI Project

In your script, you can call the RegDBDeleteValue and RegDBDeleteKey functions in your uninstallation event-handler functions to remove specified registry values and keys.

Configuring Multiple Packages for Installation Using Transaction Processing



Project *• The following project types support multiple-package installations that use transaction processing:*

- *Basic MSI*
- *InstallScript MSI*

Windows Installer 4.5 includes support for installing multiple packages using transaction processing. The packages are chained together and processed as a single transaction. If one or more of the packages in the transaction cannot be installed successfully or if the end user cancels the installation, the Windows Installer initiates rollback for all of the packages to restore the system to its earlier state.

This section of the documentation explains how to specify .msi packages that you want InstallShield to include in your installation as chained packages. This section also explains how to configure settings such as the properties that should be passed to the chained packages.



Note *• Windows Installer 4.5 includes support for installing multiple packages using transaction processing. Earlier versions do not launch any chained .msi packages.*

If you want your installation to install Windows Installer 4.5, see Adding Windows Installer Redistributables to Projects to learn how.

Overview of Multiple-Package Installations that Use Transaction Processing



Project *• The following project types support multiple-package installations that use transaction processing:*

- *Basic MSI*
- *InstallScript MSI*

When you add one or more .msi packages to your installation as chained packages and an end user runs your installation on a system that has Windows Installer 4.5, the Windows Installer can install all of the packages in a single transaction. If one or more of the packages in the transaction cannot be installed successfully or if the end user cancels the installation, the Windows Installer initiates rollback for all packages to restore the system to its earlier state.

When the Windows Installer uses transaction processing, it batches together the various script states from the .msi packages. For example, Windows Installer executes the InstallInitialize action during the Execute sequence; at this point in the sequence Windows Installer executes all of the immediate actions for all of the packages and puts all of the deferred actions in the installation script. Next, Windows Installer executes the deferred actions—but not the commit or rollback actions—in the installation script for all of the packages. In addition, Windows Installer copies the rollback actions to the rollback script. If the products for each of the packages are successfully installed up until this point, the Windows Installer runs all of the commit actions for each of the packages; otherwise, the Windows Installer executes the rollback script to return the target system to its original state.

For more information about this functionality, see Multiple-Package Installations in the Windows Installer Help Library.



Note *• Windows Installer 4.5 includes support for installing multiple packages using transaction processing. Earlier versions do not launch any chained .msi packages.*

If you want your installation to install Windows Installer 4.5, see Adding Windows Installer Redistributables to Projects to learn how.

Adding a New Chained .msi Package to Your Project



Project *• The following project types support multiple-package installations that use transaction processing:*

- *Basic MSI*
- *InstallScript MSI*

InstallShield enables you to add already built Windows Installer packages (.msi files) to your project as chained .msi packages. Windows Installer 4.5 includes support for installing the multiple packages using transaction processing. The packages are chained together and processed as a single transaction. If one or more of the packages in the transaction cannot be installed successfully or if the end user cancels the installation, the Windows Installer initiates rollback for all packages to restore the system to its earlier state.

The Chained .msi Packages area of the Releases view is where you add to your project one or more .msi packages that you want to be chained to your main installation.



Note *• Windows Installer 4.5 includes support for installing multiple packages using transaction processing. Earlier versions do not launch any chained .msi packages.*

If you want your installation to install Windows Installer 4.5, see Adding Windows Installer Redistributables to Projects to learn how.



Task ***To add an .msi package as a chained package:***

1. In the View List under **Media**, click **Releases**.
2. Right-click the **Chained .msi Packages** explorer and then click **New Chained Package**. InstallShield adds a new item with a default name.
3. Enter a new name, or right-click it later and click **Rename** to give it a new name.

   The name is not displayed at run time; it is an internal name that is used to differentiate between different chained .msi packages in the main installation.
4. Select the new chained package item. InstallShield displays its settings in the right pane.
5. For the **Installation (run-time path)** setting, click the **Browse** button. The **Browse for File** dialog box opens.

6. Select the Windows Installer package (.msi file) that you would like to add, and then click **Open**. The package that you select must be an .msi package that can be installed through **MsiInstallProduct**. Note that it cannot be an InstallScript MSI package.

InstallShield prompts you to specify whether you want to stream the .msi package—and its uncompressed files, if the .msi package is not compressed—into your product's main .msi package:

- If you specify that you want InstallShield to stream the files, InstallShield adds the name of the .msi package to the **Installation (run-time path)** setting. InstallShield also automatically adds either the file name (if it is a compressed .msi package) or the entry ***.*** (if it is an uncompressed .msi package) to the **Streamed files** box. For the *.* wild-card entry, InstallShield streams in the .msi package as well as all of the files in the same folder as the .msi package.
- If you specify that you do not want InstallShield to stream the files, InstallShield adds the following path to the **Installation (run-time path)** setting:

  [SourceDir]FileName.msi

  You can change this path if appropriate. For example, you may want to use a path such as the following one, and also clear the **Delete streamed files after installation** check box:

  [LocalAppDataFolder]{ProductCodeGUID}\FileName.msi

  In this example, the .msi package is cached on the local system, and it is available for maintenance.

  Tip *• You may not want to stream in the .msi package, since Windows Installer has limitations for the file size of .msi packages.*

7. Configure the chained .msi package settings as needed. To learn more, see Chained .msi Package Settings.

If your main installation includes several chained .msi packages, InstallShield uses the Order column in the **ISChainPackage** table of the main installation to determine the order in which the chained packages should be launched at run time. All uninstallations are processed in reverse order, from highest to lowest order number; all installations are processed in forward order, from lowest to highest order number.

Note *• If you have specified .msi packages in the Chained .msi Packages area in the Releases view, InstallShield does not build the chained packaged when you build releases for your main installation.*

Assigning Release Flags to a Chained .msi Package

Project *• The following project types support multiple-package installations that use transaction processing:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

You can assign one or more release flags to a chained .msi package that you want to exclude from certain builds. For example, if you have a chained .msi package that should be included only in a special edition of your product that contains a special add-on that requires the chained .msi package, you can flag that chained package and include it only when it is needed.

Task ***To add a release flag to a chained .msi package that you have added to your installation project:***

1. In the View List under **Media**, click **Releases**.
2. In the **Chained .msi Packages** explorer, select the package that should contain the release flag.
3. For the **Release flags** setting, type a string. The string can be any combination of letters or numbers. To have more than one flag on a chained package, use a comma to separate the flags.

To learn more about filtering chained .msi packages based on release flags, see Release Flags.

Removing a Chained .msi Package from Your Project

Project • *The following project types support multiple-package installations that use transaction processing:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

Task ***To remove a chained .msi package from your project:***

1. In the View List under **Media**, click **Releases**.
2. In the **Chained .msi Packages** explorer, right-click the chained .msi package that you want to remove from your project, and then click **Remove**.

InstallShield removes the chained .msi package from your project. Note that InstallShield does not delete the .msi package, or any of its files, from your file system.

Building, Testing, and Deploying Installations

Once you have configured the features, components, files, shortcuts, registry entries, end-user dialogs, and other elements of your installation project, you are ready to create and build a release for your installation. InstallShield lets you create multiple releases for a project and configure each release for a different media type and according to different sets of requirements. Building releases packages the content of your installation, creating a disk image that you can copy to your distribution media and distribute or deploy as needed.

Testing is an essential part of creating a reliable installation. InstallShield enables you to selectively test run just the end-user interface portion of a release. You can also run your installation by simply clicking a button in InstallShield; when you run an installation by using this method, your installation executes exactly as it would on an end user's machine. All files are transferred, shortcuts and registry entries are made, and the user interface is displayed.

Before you release your product, you may want to validate your installation project. Validating a Windows Installer project involves applying a set of internal consistency evaluator (ICE) rules to your installation project. These ICEs are designed by Microsoft to help you determine whether your resulting installation package contains a valid database that performs its actions correctly.

With the debugging tools in InstallShield, you can debug your InstallScript script or Windows Installer release to identify the sources of problems. When you debug a script, you execute your script, statement by statement, and trace the flow of control by watching the execution point as it moves through the script. You can also monitor the value of any variable in your script at any point during script execution. When you debug a Windows Installer release, InstallShield runs through each action and dialog box until it reaches your breakpoint, when it halts execution. At this point, you can view and set properties.

Creating and Building Releases

Once you have designed your project in InstallShield, you are ready to create a release that you can configure and build to distribute to end users. The release is built according to the options you set for it in the Release Wizard or in settings that are displayed in the Releases view.

Working with Product Configurations



Project • *The following project types support product configurations:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

Every release that you build is a member of a product configuration. A product configuration provides a means for grouping together releases that share similar properties, such as the product name, product code, and package code.

Product configurations also enable you to keep a history of your builds without having to set up a folder structure outside InstallShield. For example, when your product is in development, you might have many releases of the same version. You do not necessarily want to overwrite each release every time that you rebuild the product. With product configurations, you can maintain your previous build history and incorporate new releases at the same time.

In the past, you may have had to maintain this type of historical record of your builds outside InstallShield. You had to create a directory structure on a drive, label each folder manually, and then copy the release into its directory. With product configurations, all of this is done for you in InstallShield.

Task ***To create a product configuration:***

1. In the View List under **Media**, click **Releases**.
2. Right-click the **Releases** explorer and then click **New Product Configuration**.

InstallShield adds a new product configuration to the Releases explorer. InstallShield lists the product configuration settings on the General tab.

For descriptions about each of the product configuration settings, see General Tab for a Product Configuration.

Tip *• You can also create product configurations through the Release Wizard. You can specify existing product configurations from the command line (using the* `-a` *parameter), but you cannot define the settings of a release through the command line.*

Selecting the Appropriate Type of Architecture Validation for Builds

Project *• The following project types support product configurations:*

- *Basic MSI*
- *Merge Module*

As x64 Windows-based systems that do not have 32-bit Windows-on-Windows (WOW64) support become more common, you may want to perform architecture validation when building a release in InstallShield. Architecture validation enables you to detect potentially problematic cases in which your installation may try to install product files or use run-time binaries that may not match the architecture of a target system.

For example, if end users may run your installation on x64 Windows Server Core systems that do not have WOW64 support, architecture validation can help you identify any x86 product files or x86 custom action files in your installation; x86 binaries cannot be loaded on x64 target systems without WOW64 support.

In addition, if you are mixing x64 and x86 product files (or x64 and x86 custom actions) in a single project and generating separate x86 and x64 .msi packages, you can use architecture validation to identify any x64 product or custom action files in your x86 releases, since these files cannot be loaded on x86 target systems.

Architecture validation also enables you to generate pure x86 .msi packages that contain only x86 versions of the built-in InstallShield custom action DLLs, or pure x64 .msi packages that contain only x64 versions of the built-in InstallShield custom action DLLs. To enable this functionality, InstallShield uses two predefined path variables: ISRedistPlatformDependentFolder and ISRedistPlatformDependentExpressFolder. By default, the values of these path variables are subfolders that contain x86 versions of the InstallShield custom action DLLs; the subfolders are in the following directory: *`InstallShield Program Files Folder`*\Redist. InstallShield modifies the values of these path variables at build time if necessary to include the x64 versions (instead of the x86 versions) of the InstallShield custom action DLLs in releases; these files are in different subfolders in *`InstallShield Program Files Folder`*\Redist.

Overview of Architecture Validation

InstallShield offers the following levels of support for build-time architecture validation:

- **None**—Bypass build-time architecture validation.
- **Lenient**—This type of validation lets you build x86 and x64 .msi packages (as indicated by the Template Summary property), and mix x86 and x64 product files and custom action files in both of those types of packages.

  At build time, the ISRedistPlatformDependentFolder and ISRedistPlatformDependentExpressFolder path variables point to the x86 locations that contain x86 InstallShield custom action DLLs. InstallShield includes these x86 custom action DLLs in the build if your project includes support that requires these custom actions.

  Lenient validation does not trigger build errors if the architecture that the Template Summary property specifies does not match the architecture for one or more of the custom action files that are being included in the release.

  Lenient validation does not trigger build warnings if the architecture that the Template Summary property specifies does not match the architecture for one or more of the product files that are being included in the release.

  Lenient is the default option.
- **Strict**—With this type of validation, InstallShield attempts to build a pure x86 or pure x64 .msi package, depending on whether the Template Summary property specifies Intel (for x86) or x64.

  If the Template Summary property specifies Intel, the ISRedistPlatformDependentFolder and ISRedistPlatformDependentExpressFolder path variables point to the x86 locations that contain x86 InstallShield custom action DLLs. However, if the Template Summary property specifies x64 and you are using strict validation, these path variables point to the x64 locations that contain x64 InstallShield custom action DLLs. If your project includes support that requires any of these custom actions, InstallShield adds the appropriate x86 or x64 versions of the DLLs to the build.

  Strict validation may trigger build errors if the architecture that the Template Summary property specifies does not match the architecture for one or more of the custom action files that are being included in the release. Thus, during build-time validation of an x86 package, InstallShield generates a build error for each x64 custom action file in the package. During build-time validation of an x64 package, InstallShield generates a build error for each x86 custom action file in the package.

  Strict validation may trigger build warnings if the architecture that the Template Summary property specifies does not match the architecture for one or more of the product files that are being included in the release. Thus, during build-time validation of an x86 package, InstallShield generates a build warning for each x64 product file in the package. During build-time validation of an x64 package, InstallShield generates a build warning for each x86 product file in the package.



***Tip** • For information on the Template Summary property, see Using the Template Summary Property.*

For additional tips on targeting 64-bit operating systems, see Targeting 64-Bit Operating Systems with Basic MSI and InstallScript MSI Installations.

Selecting the Type of Architecture Validation

InstallShield lets you specify which type of architecture validation you want to use for each product configuration that is defined in the Releases view of your project.



Task ***To select the type of architecture validation that you want InstallShield to perform at build time:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the product configuration that you want to configure.
3. On the **General** tab, in the **Architecture Validation** setting, select the appropriate option. Available options are:
   - None
   - Lenient
   - Strict

Troubleshooting Build Errors and Warnings for Strict Architecture Validation

If you are using strict architecture validation to build an .msi package that specifies x64 for the Template Summary property, InstallShield may generate errors and warnings such as the following ones at build time:

```
error -7319: 32-bit Standard DLL Custom Action MyCustomAction must not be included in a strict 64-bit
    package.
```

```
warning -7326: Including 32-bit PE file C:\SourceFiles\x86\Myx86File.exe in a strict 64-bit package.
```

In many cases, an x64 target system can run an x86 file. However, if the x64 system does not have WOW64 support, it may not be able to run an x86 file.

Depending on your requirements, you may want to replace the x86 file or custom action that is referenced in a build error or warning with an x64 file or custom action. In other scenarios, you may want to ignore a build error or warning. For example, if a file's component has a condition that prevents the component from being installed on x64 systems, you may want to ignore its corresponding build warning. In some cases, you may decide to reexamine your requirements—that is, you may want to avoid supporting x64 target systems that do not have WOW64 support.

Similarly, you may encounter build errors and warnings if you are using strict architecture validation to build an x86 .msi package; for example:

```
error -7320: 64-bit Standard DLL Custom Action MyCustomAction must not be included in a strict 32-bit
    package.
```

```
warning -7327: Including 64-bit PE file C:\SourceFiles\x64\Myx64File.exe in a strict 32-bit package.
```

An x86 system cannot run any x64 files or custom actions. Depending on your requirements, you may choose to make changes to your project or ignore the build errors and warnings. In some cases, you may decide to reexamine your requirements.

Working with Releases

***Project** • For installation projects: Before building your installation project, ensure that you have completed designing and configuring the settings for every element in your project, including the features, components, files, shortcuts, registry entries, and end-user interface.*

For merge module projects: Before building your package, ensure that you have completed designing and configuring the settings for every element in your merge module, including components, files, shortcuts, and registry entries.

Once you have designed your project in InstallShield, you are ready to create a release that you can configure and build to distribute to end users. You can create multiple releases for a project and configure each release for a different media type and according to different sets of requirements. InstallShield offers several methods for creating and building a release:

- Use the Release Wizard. (This method is not available in Advanced UI or Suite/Advanced UI projects.)
- Use the Releases view.
- Build from the command line using **ISCmdBld.exe**.
- Build a release through the automation interface. (This method is not available in Advanced UI or Suite/Advanced UI projects.)

For information on building an InstallShield release from within Visual Studio, see Building Releases in Microsoft Visual Studio.

***Tip** • Make sure that Windows Explorer is not pointing to the **Disk1** folder or a subfolder when you build your release. If Windows Explorer is pointing to the **Disk1** folder, the build process fails and generates an error.*

Using the Release Wizard to Create and Build a Release

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The Release Wizard provides an easy way for you to build a product release and specify the settings for that release.

Task ***To launch the Release Wizard and build your installation package, do one of the following:***

- Click the **Release Wizard** button on the toolbar.
- On the **Build** menu, click **Release Wizard**.
- In the **Releases** view, right-click a release and click **Release Wizard**.

Follow the instructions in the panels of the Release Wizard. For a list of errors returned by the wizard, see Build Errors and Warnings.

Using the Releases View to Create and Build a Release

The Releases view lets you specify the settings for a release and build it. The procedure for creating and building a release depends on what project type you are using.

Creating and Building a Release in Basic MSI, InstallScript MSI, and Merge Module Projects



Task ***To create and build a release:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, right-click the product configuration that should contain the new release, and then click **New Release**. For information on how to create a new product configuration, see Working with Product Configurations. InstallShield adds a new release under the product configuration. Its settings are displayed on the tabs.
3. Configure the release's settings as appropriate.
4. Right-click the release and then click **Build**.

Creating and Building a Release in Advanced UI and Suite/Advanced UI Projects



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*



Task ***To create and build a release:***

1. In the View List under **Media**, click **Releases**.
2. Right-click the **Releases** explorer, and then click **New Release**. InstallShield adds a new release. Its settings are displayed on the tabs.
3. Configure the release's settings as appropriate.
4. Right-click the release and then click **Build**.

Creating and Building a Release in an InstallScript or InstallScript Object Project



Task ***To create and build a release:***

1. In the View List under **Media**, click **Releases**.
2. Right-click the **Releases** explorer, point to **New Release**, and click the type of media that you want to create. InstallShield adds a new release to the explorer. Its settings are displayed on the tabs.
3. Configure the release's settings as appropriate.
4. Right-click the release and then click **Build**.

Setting the Release Location

The disk image folders for your installation are built in the release location. All additional files and folders that are necessary for storing uncompressed application files are placed in subfolders of the disk image folders. The release location is a subfolder of your project's location.

To see your release after you have built it, click **Disk Images** under the product name and the release name in the Releases view. If you have not changed the release location, InstallShield places the built installation package into the following folder:

*InstallShield Project Folder**project name**product configuration**release name***DiskImages\Disk1**

You can set the release location either in the Advanced Settings panel of the Release Wizard or in the Release Location property in the Releases view.

Building a Release from the Command Line

You can build a release from the command line using **ISCmdBld.exe** for Windows Installer–based projects, for InstallScript projects, and for Advanced UI or Suite/Advanced UI projects. If your project includes InstallScript, **ISCmdBld.exe** also compiles it before building the release.

Using **ISCmdBld.exe** to build an installation can be useful if you are trying to build from a batch file. Also, building a release from the command line using **ISCmdBld.exe** will handle the conversion of .ipr and .ipo files (object project files created using InstallShield Professional) to an .ism file, in addition to building the release.



Tip *• An .ism file in either binary or XML format is accepted by the command-line build. The command-line build also works with Advanced UI and Suite/Advanced UI project files (.issuite).*

You can build a release from the command line using **ISCmdBld.exe** *with the Standalone Build. For more information, see Standalone Build.*

Using ISCmdBld.exe to Build a Release from the Command Line

ISCmdBld.exe is located by default in the InstallShield folder's System subfolder. Do not move **ISCmdBld.exe** from its installed location.

For the syntax and available command-line parameters for **ISCmdBld.exe**, see ISCmdBld.exe.

Passing Command-Line Build Parameters in an .ini File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

If you want to pass numerous parameters during a command-line build, or if you consistently pass the same parameters, it might be convenient to use an .ini file. The following statement illustrates running **ISCmdBld.exe** to build a release with parameters as specified in the **MySetup.ini** file.

```
ISCmdBld.exe -i "C:\InstallShield 2016 Projects\MySetup.ini"
```

You need to include the same information in the .ini file as you would if you were passing the parameters at the command line. There are four sections for this file:

- **[Project]**—In this section, include entries for the path to the project file (.ism), as well as the name of the product configuration. If you are building a patch, include an entry for the name of the patch configuration that you are building.
- **[Release]**—In this section, include entries for release configuration information such as the compression type (compressed or uncompressed), build flags, **Setup.exe** settings, and the release name.
- **[Mode]**—In this section, include any of the available optional entries, such as the Silent=yes entry if you want to build your release while suppressing any build errors or warning messages. This section also lets you indicate whether a log file should be created.
- **[BuildLocation]**—In this section, you can optionally specify the release output location.

Not all sections are required. As with passing parameters directly from the command line, parameters for requirements such as silent build and build location are optional. In the example .ini file below, these parameters are in the `[Mode]` and `[BuildLocation]` sections. You can omit these entries from your .ini file if you want to accept the defaults. By default, no log file is created, the installation is not run in silent mode, and your release is created in the project location that is specified on the File Locations tab of the Options dialog box.

Sample .ini File

The following tables contain sample entries from each of the four sections in a sample .ini file. The first column of each table shows a sample entry. The other columns provide the corresponding command-line parameter and description.

Entries in the [Project] Section

Table 4-1 • Sample Entries in the [Project] Section of the .ini File

Entry	Corresponding ISCmdBld.exe Command-Line Parameter	Description
Name="C:\InstallShield 2016 Projects\Othello.ism"	-p	Path to the .ism file.
BuildLabel=Label1	-a	Name of the product configuration for the release that you are building. If it does not exist, it is created.
Product=Othello		Name of your product. This entry lets you override the value that is specified in the General Information view.
PatchConfigName="Version 1.2"	-patch_config	Name of the patch configuration in the Patch Design view that you want to build.
CompileScript=no	-n	Ensures that **Setup.rul** is not compiled as part of the build process. ***Note*** • `CompileOnly` *should not be used with* `CompileScript`.

Entries in the [Release] Section

Table 4-2 • Sample Entries in the [Release] Section of the .ini File

Entry	Corresponding ISCmdBld.exe Command-Line Parameter	Description
BuildFlags=flags	-f	Release flags to be included in the installation, separated by commas.
Configuration=COMP	-c	Compressed vs. uncompressed.
Name=Othello Beta	-r	Release name.
SetupEXE=yes	-e	Creates a **Setup.exe** file.

Entries in the [Mode] Section

Table 4-3 • Sample Entries in the [Mode] Section of the .ini File

Entry	Corresponding ISCmdBld.exe Command-Line Parameter	Description
CubFile="C:\Program Files\InstallShield\2016\Support\Validation\MyValidation.cub"	-m	Validate the installation or merge module package after it is built using the specified .cub file. Include the full path and name to the .cub file. To use multiple .cub files, separate each path with a semicolon (;). Enclose long file names in quotation marks. For example, the following entry indicates that validation should be performed with the InstallShield Validation Suite for Windows 8 (**ISWin8.cub**) and the Full MSI Validation Suite (**darice.cub**): `CubFile="C:\Program Files\InstallShield\2016\Support\Validation\ISWin8.cub;C:\Program Files\InstallShield\2016\Support\Validation\darice.cub"`
Silent=yes	-s	Runs in silent mode.
StopOnFirstError=yes	-x	Aborts the build when the first error is encountered.
CompileOnly=no	-q3	Compiles only **Setup.rul** and streams **Setup.inx** into the **Binary** table of the .msi file, if one was previously built.  ***Note*** • *The entries* `CompileOnly`, `BuildTablesRefreshFiles`, *and* `BuildTablesOnly` *are mutually exclusive. Only one can be set to* **yes**. `CompileOnly` *should not be used with* `CompileScript`.
BuildTablesRefreshFiles=no	-q2	Builds the Windows Installer tables and refreshes files.  ***Note*** • *The entries* `CompileOnly`, `BuildTablesRefreshFiles`, *and* `BuildTablesOnly` *are mutually exclusive. Only one can be set to* **yes**.

Table 4-3 • Sample Entries in the [Mode] Section of the .ini File (cont.)

Entry	Corresponding ISCmdBld.exe Command-Line Parameter	Description
BuildTablesOnly=yes	-q1	Builds only the Windows Installer tables for your release.  ***Note*** • *The entries* `CompileOnly`, `BuildTablesRefreshFiles`, *and* `BuildTablesOnly` *are mutually exclusive. Only one can be set to* **yes**.
UpgradeOnly=no	-u	Allows you to upgrade—but not build—your release.
Verbose=yes	-v	Generates an engine log file.
WarningAsError=yes	-w	Treats any build warnings as build errors.
MSIVersion=2.0	-g	Version of Windows Installer required on target system.  ***Note*** • *This entry applies to the Standalone Build only.*
DotNetVersion=	-j	Minimum version of Microsoft .NET Framework required on the target system. This parameter is optional and defaults to the latest version of the .NET Framework supported by InstallShield.  ***Note*** • *This entry applies to the Standalone Build only.*
DotNetPath="C:\..."	-t	Path to the Microsoft .NET Framework on the build machine.  ***Note*** • *This entry applies to the Standalone Build only.*

Table 4-3 • Sample Entries in the [Mode] Section of the .ini File (cont.)

Entry	Corresponding ISCmdBld.exe Command-Line Parameter	Description
SkipValidators=yes	-h	Enables you to turn off the upgrade validators that normally run at the end of the build.  ***Note*** • *This entry applies to the Standalone Build only.*
MergeModulePath="C:\..."	-o	Search path for merge modules (.msm files). For more information, see Specifying the Directories that Contain Merge Modules.  ***Note*** • *This entry applies to the Standalone Build only.*
PrerequisitePath="C:\..."	-prqpath	Search path for InstallShield prerequisite files (.prq). For more information, see Specifying the Directories that Contain InstallShield Prerequisites.  ***Note*** • *This entry applies to the Standalone Build only.*

Entries in the [BuildLocation] Section

Table 4-4 • Sample Entries in the [BuildLocation] Section of the .ini File

Entry	Corresponding ISCmdBld.exe Command-Line Parameter	Description
Path="C:\InstallShield 2016 Projects\Othello"	-b	Release output location.

For more information about the .ini file entries, see Building a Release from the Command Line.

Using ISBuild.exe to Build from the Command Line



Project • *This information applies to InstallScript projects.*

ISBuild.exe, a command-line tool that is available for legacy InstallScript projects, is a shell that calls into **ISCmdBld.exe**. **ISCmdBld.exe** handles the conversion of .ipr and .ipo (object projects created using InstallShield Professional) files to an .ism file and builds the release.

For a list of command-line parameters for **ISBuild.exe**, see ISBuild.exe.

Building a Self-Extracting Executable File from the Command Line



Project • *This information applies to InstallScript projects.*

You can build a self-extracting executable file from the command line using **ReleasePackager.exe**. This can be useful if you are building from a batch file.

ReleasePackager.exe is located by default in the InstallShield folder's **System** subfolder. Do not move **ReleasePackager.exe** from its installed location.

For syntax and available arguments and options for **ReleasePackager.exe**, see ReleasePackager.exe.

Rebuilding Releases



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

You can rebuild a release at any time using the default entries that you already provided in the settings in the Releases view. The release that currently has focus is rebuilt.



Task ***To rebuild a release, do one of the following:***

- Click the **Build** button.
- On the **Build** menu, click **Build**.
- In the **Releases** view, right-click a release name and then click **Build**.

You can run the Release Wizard again to rebuild a release. Because the wizard stores the release settings in your InstallShield project file, the wizard displays all of the options that you set when you last built the release, even if you deleted the release.

To see your new installation package, click Disk Image(s) for the appropriate release in the Releases view. InstallShield places the built installation package into the following folder if you have not changed the default release location:

C:\InstallShield 2016 Projects*project nameproduct configuration**release name*\DiskImages\Disk1**

***Note** • Make sure that Windows Explorer is not pointing to the **Disk1** folder or a subfolder when you build your release. If it is pointing to the **Disk1** folder, the build process can continue indefinitely. If Windows Explorer is accessing a subfolder, the build generates an error.*

Performing Quick Builds

When you are testing your installation, you might not want to continually build all of your installation files if no files have been changed. InstallShield provides you with two quick build options: Build Tables Only and Build Tables & Refresh Files.

Build Tables Only

As the name of this option implies, only the Windows Installer tables are built. If you have not built this installation already, a new .msi file is created, but no files are added to your installation. If you have built your installation already, the .msi file is updated when all the tables are built, but no files are transferred. You can use this option to test the installation's user interface.

Task ***To build only your installation's Windows Installer tables, do one of the following:***

- On the **Build** menu, click **Build Tables Only**.
- In the **Releases** view, right-click the release that you would like to build and click **Build Tables Only**.

Build Tables & Refresh Files

If you would like a near-complete build without having to transfer all of the installation's files to a new location, you can perform a build that rebuilds your .msi file and updates the **Files** table, thereby including any new or changed files in your installation. Changed files are updated only if the size or time stamp differs from the copy already included in the build. References to deleted files are removed from the installation, but the file remains in the build location. This type of build works only when the media is an uncompressed network image.

Task ***To build your Windows Installer tables and refresh the installation's files, do one of the following:***

- On the **Build** menu, click **Build Tables & Refresh Files**.
- In the **Releases** view, right-click the release that you would like to build and click **Build Tables & Refresh Files**.

Performing Batch Builds

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

- *InstallScript Object*
- *Merge Module*

You might need to build multiple releases with different configurations simultaneously. This functionality is known as a batch build.



Task ***To perform a batch build through InstallShield:***

1. In the View List under **Media**, click **Releases**.
2. In a Basic MSI, InstallScript MSI, or Merge Module project: Right-click a product configuration and click **Batch Build**.

   In an InstallScript or InstallScript Object project: Right-click the **Releases** explorer and click **Batch Build**.

The Batch Build dialog box opens to prompt you to select the product configurations and releases that you would like to build.

Specifying Commands that Run Before, During, and After Builds



Edition • *This functionality is available in the Premier edition of InstallShield.*



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *Suite/Advanced UI*

Some project-specific differences are noted where applicable.

InstallShield lets you specify commands that you want to be run at various stages of the build process. For example, if you are developing a Basic MSI project, you may want to modify the .msi package that InstallShield creates at build time before InstallShield digitally signs it and streams it into the **Setup.exe** file. You can specify a command that runs a script that makes the required changes to the .msi package at the appropriate time during the build.

When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use the following variables that are defined specifically for build event commands.

Table 4-5 • Build Event Variables

Variable	Project Types	Description
<ISReleaseName>	Basic MSI, InstallScript, InstallScript MSI, Merge Module, Suite/Advanced UI	This variable indicates the name of the release that InstallShield is building.
<ISProductConfigName>	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This variable indicates the name of the product configuration that contains the release that InstallShield is building. (Note that this is always *Media* for InstallScript projects.)
<ISReleasePath>	Basic MSI, InstallScript, InstallScript MSI, Merge Module, Suite/Advanced UI	This variable indicates the build location of the release that InstallShield is building. It is set through the Release Location setting on the Build tab for a release in the Releases view.
<ISReleaseUsesShallowFolderPaths>	Basic MSI, InstallScript MSI	This variable is set to true or false to indicate whether InstallShield uses a shallow folder directory structure when building your release.

At build time, InstallShield sets temporary `<ISReleaseName>`, `<ISProductConfigName>`, `<ISReleasePath>`, and `<ISReleaseUsesShallowFolderPaths>` environment variables; InstallShield also sets temporary environment variables for all of the defined path variables in your project. You can refer to the value of one of these environment variables in a batch file by surrounding the variable name with percent signs (%); for example:

set PATH = %<ISReleaseName>%;%PATH%

When the build is complete, InstallShield deletes the temporary environment variables that it set.



Task ***To specify one or more commands that run before, during, or after a build:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the release that you would like to configure.
3. Click the **Events** tab.
4. In one or more of the following settings, enter the commands that you want InstallShield to run:
   - Prebuild Event

- Precompression Event (This event is not available in Suite/Advanced UI projects.)
- Postbuild Event

Enter each command as it would be launched from the command prompt. Thus, for example, if you specify a path that includes spaces, enclose the path within quotation marks.

For details about each setting, see Events Tab for a Release.

To specify more than one command for any of those settings, click the ellipsis button (**...**) in the setting. A dialog box opens, enabling you to enter one or more commands. Enter each command on a separate line.

If you enter more than one command for an event settings, InstallShield runs each command at build time in the order that they are listed for that setting. The build waits until a command finishes before proceeding to the next one.



***Tip** • A verbose build log shows each of the build events that are run at build time. To troubleshoot build events, generate a verbose log file and search for strings such as "Launching prebuild events", "Launching precompression build events", and "Launching postbuild events". These areas of the log list details about the commands that are run at build time.*

To generate a verbose build log, pass the `-v` *parameter when launching* **IsCmdBld.exe** *from the command line.*

Resolving Build Errors and Warnings

When you build a release, the Output window opens across the bottom of the InstallShield user interface. The Tasks tab of this window lists any build errors and warnings that occur during the build process.



Task ***To learn how to resolve a build error or warning that you encounter, do one of the following:***

- On the **Tasks** tab of the **Output** window, click an error code to launch your Internet browser and see Knowledge Base articles and HelpNet topics for the selected build error or warning.
- See Build Errors and Warnings, which provides tips on how to resolve build errors and warnings.

Changing Project Settings from the Command Line

The InstallShield automation interface enables you to query and modify many project settings from an unattended build process, using, for example, a VBScript script or Visual Basic application.

The framework of any script that uses the automation interface to access a project appears as follows (you can copy this script into a text file called **Framework.vbs** and then double-click the file icon):

```
Set oProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
oProject.OpenProject "C:\MySetups\MyProject.ism"

' perform queries and changes here

oProject.SaveProject ' necessary only if modifying the project
oProject.CloseProject
```

Canceling Builds

Task ***To cancel a build after it has started:***

Click the **Stop Build** button on the toolbar.

Standalone Build

InstallShield provides a Standalone Build module that enables you to install only the part of InstallShield that compiles the installations, plus any redistributables that you want to include. For instructions on installing the Standalone Build and any redistributables on a build machine, see Installing the Standalone Build on a Build Machine.

The InstallShield 2016 Standalone Build can coexist with other major versions of the Standalone Build on the same machine (in different folders).

Upgrading Projects

The Standalone Build automatically upgrades projects that were created with InstallShield Developer 7.0 or later. However, it does not upgrade projects that were created with InstallShield for Windows Installer.

Using the Command-Line Build

You can run the Standalone Build from the command line. For more information, see Standalone Command-Line Build.

Using the Standalone Automation Interface

The Standalone Build includes a standalone version of the automation interface—the Standalone Automation Interface.

Installing the Standalone Build on a Build Machine

Edition • *Support for multilanguage installations is available with InstallShield Premier Edition only.*

Installing the Standalone Build

The installation for the Standalone Build consists of a single, compressed executable file. It includes all of the files that are required for building multilanguage installations.

Task ***To install the Standalone Build on a build machine:***

1. Locate the executable file for the Standalone Build installation:
   - If you have the InstallShield DVD, the file is on the DVD and you can find it using the DVD Browser.

- If you downloaded InstallShield, the Standalone Build installation file is available for download as documented in the InstallShield download and license instructions (http://www.installshield.com/instructions/sab.asp).

2. Double-click the executable file and run the installation. For detailed information on configuring licensing for the Standalone Build, see the InstallShield download and license instructions (http://www.installshield.com/instructions/sab.asp).

During the installation, the Custom Setup dialog shows two Standalone Build features that are disabled by default. Enable these features if you want to be able to use them.

- **Automation Interface**—This feature installs and registers the files that you need to use the automation interface with the Standalone Build.
- **InstallScript Objects Support**—This feature installs and registers the InstallScript Object support files that enable you to build InstallScript projects that contain InstallScript Objects.

Files that Are Registered During the Standalone Build Installation

When you install the Standalone Build, the installation installs and registers each of the following merge modules if they are not present:

- ATL 3.0 (**ATL.msm**)
- Microsoft C Runtime Library (**MSVCRT.msm**)
- Microsoft C Runtime Library 7.1 (**VC_User_CRT71_RTL_X86_---.msm**)
- MSXML3 (**msxml3_wim32.msm**)
- MSXML4 (**msxml4sxs32.msm** or **msxml4sys32.msm**)
- Visual C++ 8.0 ATL (x86) WinSXS MSM (**Microsoft_VC80_ATL_x86.msm**)
- Visual C++ 8.0 ATL.Policy (x86) WinSXS MSM (**policy_8_0_Microsoft_VC80_ATL_x86.msm**)
- Visual C++ 8.0 CRT (x86) WinSXS MSM (**Microsoft_VC80_CRT_x86.msm**)
- Visual C++ 8.0 CRT.Policy (x86) WinSXS MSM (**policy_8_0_Microsoft_VC80_CRT_x86.msm**)

The Standalone Build installation also installs and registers the Microsoft Scripting Runtime Library (**scrrun.dll**) if it is not present. It is installed to the [SystemFolder].

If you copy the Standalone Build files from one machine to another instead of running the installation on that second machine, you must manually install the technologies that are included in these merge modules and the **scrrun.dll** file if they are not already present. You can obtain the installation packages for the merge module technologies from the Microsoft Web site (http://www.microsoft.com).

Files that Are Registered During the Installation of the Optional Automation Support Feature

The Automation Interface feature in the Standalone Build installation installs and registers the files that you need in order to use the automation interface with the Standalone Build. Following is the list of files that need to be registered:

- *Program Files Folder***Common Files\InstallShield\Shared\IsmAuto.tlb**
- *Program Files Folder***Common Files\InstallShield\Shared\Ismmupdater.tlb**
- *Program Files Folder***Common Files\InstallShield\Shared\ISWIBuild.tlb**

- *Program Files Folder***Common Files\InstallShield\Shared\IsAppServices.tlb**
- *Program Files Folder***Common Files\InstallShield\Shared\IsUpgrade.tlb**
- *Standalone Build Program Files Folder***System\ISWiAutomationAutomation Interface Version.dll**

If you copy the Standalone Build files from one machine to another instead of running the installation on that second machine, and if you want to be able to use the automation interface with the Standalone Build, you must manually register these files.

Files that Are Registered During the Installation of the Optional InstallScript Objects Support Feature

The InstallScript Objects Support feature installs and registers the InstallScript Object support files that enable you to build InstallScript projects that contain InstallScript Objects. Following is the list of files that need to be registered:

- *Standalone Build Program Files Folder***System\DObjAutomation Interface Version.dll**
- *Standalone Build Program Files Folder***System\ISBE.dll**
- *Standalone Build Program Files Folder***System\ISMKAutomation Interface Version.dll**
- *Standalone Build Program Files Folder***System\StockWiz.dll**

If you copy the Standalone Build files from one machine to another instead of running the installation on that second machine, and if you want to be able to build InstallScript projects that contain InstallScript Objects, you must manually register these files.

Manually Registering Files for the Standalone Build

To manually register one of the aforementioned DLL files for the Standalone Build, use **Regsvr32.exe**, which is installed in the System32 folder.

To manually register one of the .tlb files, use the following file:

*Standalone Build Program Files Folder***System\RegTypLib.exe**

Adding Redistributables to the Build Machine for the Standalone Build

Most of the redistributables—including merge modules, objects, and InstallShield prerequisites—are not installed automatically with the Standalone Build because they would require significantly more hard disk space. This allows you to install on the build machine only the redistributables that are required by your InstallShield projects. The only one that is included is the .NET Framework 2.0 redistributable.

Tip *• The Standalone Build uses the same basic directory structure that InstallShield uses for its program files. Therefore, if you need to copy a redistributable or some other file from a machine that has InstallShield to a machine that has the Standalone Build, use the same relative path. For example, if a file is in the* ***InstallShield*** **Program Files Folder\Modules\i386** *directory on the machine that has InstallShield, copy that file to the* **Standalone Build Program Files Folder\Modules\i386** *directory on the machine that has the Standalone Build.*

Task ***To add a .NET redistributable to your build machine:***

1. Locate the required redistributables on the machine that has the full version of InstallShield. The file is typically located in **InstallShield Program Files Folder\Redist** or a subfolder of the Redist folder. If an InstallShield

redistributable is not available on your machine, you must download it first. For more information, see Redistributable Downloader Wizard.

2. Copy the redistributable from the machine that has InstallShield to the Standalone Build Program Files Folder on the build machine.



Task ***To add a redistributable for Basic MSI or InstallScript MSI projects to your build machine:***

1. Locate the required redistributables on the machine that has the full version of InstallShield. If an InstallShield redistributable is not available on your machine, you must download it first. For more information, see Downloading Redistributables to Your Computer.

   - The two default locations for InstallShield merge modules are:

     *InstallShield Program Files Folder***Modules\i386**

     Common Files\Merge Modules

   - The default location for the InstallShield objects is:

     *InstallShield Program Files Folder***Objects**

   - All InstallShield prerequisites are stored in the following location:

     *InstallShield Program Files Folder***SetupPrerequisites**

   - If you built your own merge modules in InstallShield, you would find them in any of the locations that are listed on the Merge Modules tab of the Options dialog box.

2. Copy the redistributable from the machine that has InstallShield to appropriate location on the build machine.

   - Add merge modules to the following folder:

     *Standalone Build Program Files Folder***Modules\i386**

   - Add objects to the following folder:

     *Standalone Build Program Files Folder***Objects**

   - Add InstallShield prerequisites to the following folder:

     *Standalone Build Program Files Folder***SetupPrerequisites**



Task ***To add an InstallShield prerequisite for InstallScript projects to your build machine:***

1. Locate the required InstallShield prerequisite on the machine that has the full version of InstallShield. If an InstallShield redistributable is not available on your machine, you must download it first. For more information, see Downloading Redistributables to Your Computer.

   All InstallShield prerequisites are stored in the following location:

   *InstallShield Program Files Folder***SetupPrerequisites**

2. Copy the InstallShield prerequisite from the machine that has InstallShield to appropriate location on the build machine:

   *Standalone Build Program Files Folder***SetupPrerequisites**



Task ***To add to your build machine one of the other InstallShield redistributables for InstallScript projects:***

1. Download the installation for the redistributable. The installations are available through FlexNet Connect. For more information, see Obtaining Updates for InstallShield.
2. Run the redistributable installation on the build machine. When the InstallShield Wizard displays the Choose Destination Location dialog, browse to the following location:

   *Standalone Build Program Files Folder***ObjectsPro**

   If the ObjectsPro folder has not been created yet, you will need to add that folder.

The InstallScript object installations install and register the required object files.



Task ***To add to your build machine one of your own InstallScript redistributables or a third-party redistributable:***

1. Locate the redistributable.
2. Place it in the following location on the build machine:

   *Standalone Build Program Files Folder***ObjectsPro**

   If the ObjectsPro folder has not been created yet, you will need to add that folder.

Standalone Command-Line Build

The Standalone Build lets you use **ISCmdBld.exe** to build releases from the command line. For instructions, see Building a Release from the Command Line.

For a list of command-line parameters that **ISCmdBld.exe** supports, see ISCmdBld.exe.

Standalone Automation Interface

The Standalone Build includes a standalone version of the automation interface—the InstallShield Standalone Automation Interface. This interface uses the same **ISWiAutomationAutomation Interface Version.dll** file that is installed with InstallShield, but it is installed to a different location.



Note • *If you install the Standalone Build on the same machine as InstallShield, the first* **ISWiAutomation*Automation Interface Version*.dll** *file that is installed is the one that is registered. To learn more, see Installing the Standalone Build and InstallShield on the Same Machine.*

Future versions of the Standalone Build will have new ProgIDs and new GUIDs. This enables you to install multiple versions of the Standalone Automation Interface side by side on the same build machine.

Registering ISWiAutomationAutomation Interface Version.dll

The Standalone Automation Interface is contained in the file **ISWiAutomationAutomation Interface Version.dll**. If you installed the Standalone Build by running the installation for the Standalone Build, this file is already registered. If you copied the files for the Standalone Build instead of running the installation, you need to register this file manually using **Regsvr32.exe**, which is installed in the System32 folder. The **ISWiAutomationAutomation Interface Version.dll** file is installed in the System subfolder of the Standalone Build Program Files folder.

Installing the Standalone Build and InstallShield on the Same Machine

In most cases, the Standalone Build is not installed on the same machine where InstallShield is installed. If you do install both on the same machine, you need to be aware that the Standalone Automation Interface uses the same **ISWiAutomationAutomation Interface Version.dll** file that InstallShield uses, but it is installed to a different location. In addition, both the Standalone Automation Interface and InstallShield use the same ProgID: **IswiAutoAutomation Interface Version.ISWiProject**. This enables you to use the same automation script with both the Standalone Automation Interface and InstallShield, without requiring you to change your code to reflect the appropriate ProgID.

When you first install either InstallShield or the Standalone Build, the installation registers **ISWiAutomationAutomation Interface Version.dll**. If you next install the other tool, that second installation detects that **ISWiAutomationAutomation Interface Version.dll** is already registered, so it does not re-register it. As long as one instance of **ISWiAutomationAutomation Interface Version.dll** is registered, you can use both the Standalone Build and InstallShield with the automation interface.

If you uninstall the first instance that you originally installed, the installation unregisters **ISWiAutomationAutomation Interface Version.dll**. Therefore, you must manually register **ISWiAutomationAutomation Interface Version.dll** so that you can use the automation interface with the product that still remains. For example, if you install InstallShield, install the Standalone Build, and then uninstall InstallShield, you must next manually register the **ISWiAutomationAutomation Interface Version.dll** file for the Standalone Build; otherwise, you cannot use the Standalone Automation Interface.

To manually register **ISWiAutomationAutomation Interface Version.dll**, use **Regsvr32.exe**, which is installed in the System32 folder. The **ISWiAutomationAutomation Interface Version.dll** file is installed in the System subfolder of the InstallShield and Standalone Build Program Files folders.

Microsoft Build Engine (MSBuild)

InstallShield supports the Microsoft Build engine (MSBuild) included with the .NET Framework. MSBuild support enables you to build Visual Studio solutions with InstallShield projects in build lab environments where Visual Studio is not installed.

Overview

MSBuild is an extensible build framework designed to remove the build dependence on Visual Studio. The .NET Framework functions in a role similar to the InstallShield Standalone Build, providing the capability to build projects or solutions from the command line or any other host of MSBuild. For more information on MSBuild, see the MSDN Library.

MSBuild Tasks

The flexibility and extensibility of MSBuild is controlled through atomic groupings of internal build steps referred to as *tasks*. One example of a task that ships with MSBuild is Csc, which can compile code from a Visual C# project. InstallShield installs an MSBuild task called InstallShield, which builds an InstallShield project, and a targets file that provides default build steps for the project. This customized task, along with the targets file, enables MSBuild to perform all required actions to build the InstallShield project as part of a Visual Studio solution.

This table describes the parameters of the InstallShield task. The Project Type column lists the applicable project types

Table 4-6 • MSBuild InstallShield Task

Parameter	Type	Project Type	Description
InstallShieldPath	String	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This parameter specifies the path to folder containing the InstallShield application.
Project	String	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This parameter specifies the location of the project file (.ism).
ProductConfiguration	String	Basic MSI, InstallScript MSI, Merge Module	This parameter specifies the product configuration for the release.
ReleaseConfiguration	String	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This parameter specifies the release name. ***Note*** • *You must specify the ReleaseConfiguration or the PatchConfiguration. If you do not specify one of these, or if you specify both of these, a build error occurs.*
PatchConfiguration	String	Basic MSI, InstallScript MSI	This parameter specifies the name of the patch configuration in the Patch Design view that is being built. ***Note*** • *You must specify the ReleaseConfiguration or the PatchConfiguration. If you do not specify one of these, or if you specify both of these, a build error occurs.*

Table 4-6 • MSBuild InstallShield Task

Parameter	Type	Project Type	Description
OutDir	String	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This parameter specifies the fully qualified path to the folder where you want the output folders and files to be placed.
MergeModulePath	String[ ]	Basic MSI, InstallScript, InstallScript MSI	This parameter specifies one or more folders that contain the merge modules (.msm files) that are referenced by your project. InstallShield provides additional ways for specifying the folders that contain merge modules. For more information, see Specifying the Directories that Contain Merge Modules.
PrerequisitePath	String[ ]	Basic MSI, InstallScript, InstallScript MSI	This parameter specifies one or more semicolon-delimited folders that contain the InstallShield prerequisite files (.prq files) that are referenced by your project. InstallShield provides additional ways for specifying the folders that contain InstallShield prerequisite files. For more information, see Specifying the Directories that Contain InstallShield Prerequisites.
ReleaseFlags	String[ ]	Basic MSI, InstallScript MSI	This parameter enables you to specify any release flags that you want to include in your release. Separate multiple flags with commas.
PathVariables	ITaskItem[ ]	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This parameter enables you to override the path variables for the project. Using this parameter, you can specify the path to the path variable and also define a value for the name of the path variable using the PathVariable subelement. To reference a file in a sibling project that is within the same Visual Studio solution folder, use the predefined path variable called VSSolutionFolder. For more information, see Using the VSSolutionFolder Path Variable with Visual Studio Solutions. For more information on MSBuild ITaskItem[ ] properties, see the MSDN Library.

Table 4-6 • MSBuild InstallShield Task

Parameter	Type	Project Type	Description
PreprocessorDefines	ITaskItem[ ]	Basic MSI, InstallScript, InstallScript MSI	This parameter enables you to add or override the preprocessor defines for the project. Using this parameter, you can specify the definition of the preprocessor define and also define a value for the name of the preprocessor define using the Token subelement. For more information on MSBuild ITaskItem[ ] properties, see the MSDN Library.
OutputGroups	ITaskItem[ ]	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This parameter specifies the project output groups from the Visual Studio solution. Using this parameter, you can specify the path to the source location of the project output group and also define these additional values using these subelements as listed: • **Name**—Name of the project • **OutputGroup**—Name of the project output group • **TargetPath**—Target path of the project output group (different from its source location) For more information on MSBuild ITaskItem[ ] properties, see the MSDN Library.

Table 4-6 • MSBuild InstallShield Task

Parameter	Type	Project Type	Description
Build	String	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This parameter specifies what type of build to perform. You can choose from these types of builds: • **Complete**—This option builds the entire release. • **Compile**—This option compiles the **Setup.rul** script file. In Basic MSI and Merge Module projects, this option also streams **ISSetup.dll** into the **Binary** table of the .msi package. • **Tables**—This option builds only the Windows Installer tables for the database. If you have not built this installation already, a new .msi file is created, but no files are added to your installation. If you have built your installation already, the .msi file is updated when all the tables are built, but, again no files are transferred. Ideally, this option is to be used to test the user interface of your installation. This option is applicable to Basic MSI, InstallScript MSI, Merge Module projects. • **TablesAndFiles**—For Basic MSI, InstallScript MSI, and Merge Module projects, this option builds the Windows Installer tables and refreshes the files. This type of build can be run only after a complete build has been performed, and it works only when the media is an uncompressed network image. For InstallScript projects, this option rebuilds only those portions of the release that have changed since the last build. If this parameter is not used, the entire file media library is rebuilt. • **UpgradeOnly**—This option enables you to upgrade—but not build—a Basic MSI project that was created in an earlier version of InstallShield.
BuildCompressed	Boolean	Basic MSI, InstallScript MSI, Merge Module	This parameter specifies whether your release is compressed into one file or remains uncompressed in multiple files.

Table 4-6 • MSBuild InstallShield Task

Parameter	Type	Project Type	Description
BuildSetupExe	Boolean	Basic MSI, InstallScript MSI, Merge Module	For Basic MSI and InstallScript MSI projects, this parameter specifies whether you want to create a **Setup.exe** file along with your installation. For merge module projects, this option specifies whether you want to build the merge module and also copy it to the merge modules folder, or just build the .merge module without copying it.
ProductVersion	String	Basic MSI, InstallScript MSI, Merge Module	This parameter specifies the product version. This is especially helpful if you want to increment the build version (the third field) of the product version. For information on valid product version numbers, see Specifying the Product Version.
PropertyOverrides	ITaskItem[ ]	Basic MSI, InstallScript MSI, Merge Module	This parameter enables you to override the value of a Windows Installer property or create the property if it does not exist. To use this parameter, include a property list of items whose value is the new property value, and whose metadata `Property` is the name of the property. This parameter is exposed as the InstallShieldPropertyOverrides ItemGroup passthrough to the PropertyOverrides property on the InstallShield task.
RunMsiValidator	ITaskItem[ ]	Basic MSI, InstallScript MSI, Merge Module	This parameter enables you to specify one or more .cub files to use for validating the installation package or merge module after it is built. To learn more about validation, see Validating Projects.
RunUpgradeValidation	Boolean	Basic MSI, InstallScript MSI	This parameter specifies whether or not to run upgrade validation.
StopOnFirstError	Boolean	Basic MSI, InstallScript MSI, Merge Module	This parameter specifies whether or not to stop the build after an initial error.
MsiVersion	String	Basic MSI, InstallScript MSI, Merge Module	This parameter specifies the minimum version of Windows Installer that the installation can accept on the target machine.

Table 4-6 • MSBuild InstallShield Task

Parameter	Type	Project Type	Description
DotNetFrameworkVersion	String	Basic MSI, InstallScript MSI	This parameter specifies the minimum version of the .NET Framework that the installation can accept on the target machine.
DotNetUtilPath	String	Basic MSI, InstallScript MSI	**Regasm.exe** and **InstallUtilLib.dll** are utilities that are included with each version of the .NET Framework. This parameter specifies the path for the directory that contains the 32-bit version of these files that you want to use at build time for releases that include .NET installer classes and COM interop. This is not the path to .NET Framework redistributable files. ***Note*** • *If you are using InstallShield on a 64-bit system,* **ISCmdBld.exe** *determines the location of the corresponding 64-bit version of the .NET Framework based on the path that you specify for this parameter, and* **ISCmdBld.exe** *uses the 64-bit location of* **Regasm.exe** *and* **InstallUtilLib.dll** *when appropriate.*
Disk1Folder	Output String	Basic MSI, InstallScript, InstallScript MSI, Merge Module	This parameter specifies the location of the output folder.

MSBuild Scripts

MSBuild natively understands one file format: its XML build script, which is used as the project file format for Visual C# and Visual Basic .NET projects (.csproj and .vbproj). MSBuild also has internal hooks to handle solution files and the Visual C++ project file format (.sln and .vcproj).

The InstallShield integration with Visual Studio uses an MSBuild-compatible XML format project file (.isproj), which enables MSBuild to seamlessly build Visual Studio solutions that include InstallShield projects. To build solutions in a standalone environment, install the InstallShield Standalone Build on the build machine.

Customizing the .isproj File

To incorporate changes for your InstallShield project into your .isproj file, add a PropertyGroup element or an ItemGroup element near the top of your .isproj file, or update an existing PropertyGroup or ItemGroup element. Then add InstallShield-related parameters as needed.

The following sample code from an .isproj file demonstrates how to do the following:

- Set the product version.

- Set the product name.
- Set a custom public property called **MY_PROPERTY** to the value *My Value*.
- Override a path variable called **MyPathVariableName** with a new value of *C:\MyPath*.
- Specify the following search paths for InstallShield prerequisites: **<ISProductFolder>\SetupPrerequisites** and **<ISProjectFolder>\MyCustomPrerequisites**.

```
<PropertyGroup>
   <InstallShieldProductVersion>1.2.3</InstallShieldProductVersion>
</PropertyGroup>
<ItemGroup>
   <InstallShieldPropertyOverrides Include="My New Product">
       <Property>ProductName</Property>
   </InstallShieldPropertyOverrides>
   <InstallShieldPropertyOverrides Include="My Value">
       <Property>MY_PROPERTY</Property>
   </InstallShieldPropertyOverrides>
   <InstallShieldPathVariableOverrides Include="C:\MyPath">
      <PathVariable>MyPathVariableName</PathVariable>
   </InstallShieldPathVariableOverrides>
   <InstallShieldPrerequisitePath Include="&lt;ISProductFolder&gt;\SetupPrerequisites"/>
   <InstallShieldPrerequisitePath Include="&lt;ISProjectFolder&gt;\MyCustomPrerequisites"/>
</ItemGroup>
```

Using MSBuild to Build a Release from the Command Line

***Note** • If you use MSBuild to build Visual Studio solutions with InstallShield projects, MSBuild requires .NET Framework 3.5 or later.*

MSBuild provides an easy way to build a release from the command line on a machine on which Visual Studio is not installed. The only components you must have installed on the machine are the .NET Framework and the InstallShield Standalone Build. Place a copy of your Visual Studio solution on the machine, and run MSBuild.

Task ***To use MSBuild from the command line:***

1. Open the Command Prompt window.
2. Change the directory to the one that contains **MSBuild.exe**:

   ```
   C:\Windows\Microsoft.NET\Framework\Version Folder\
   ```

3. Type the command-line statement to build the release build of the Visual Studio integration project. For example:

   ```
   MSBuild.exe C:\Folder Containing My Visual Studio Solution\My Solution.sln /
       property:Configuration=Release
   ```

Configuring Release Settings

When you create a release, it has default settings. You can edit all of the release settings in the Releases view. When you use the Release Wizard, you can also edit the settings of a release.

Task ***To edit the settings of a release:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the release whose settings you would like to configure. The settings are displayed on tabs in the right pane.
3. Select a setting on one of the tabs to modify its value. Information about the setting is displayed in the help pane or when you press F1 from within the **Releases** view.

Creating a Single Executable File for Distribution

Project • *This information applies to InstallScript projects.*

To package a build as a single executable file, use the **Create a single file executable** option in the General Options panel of the Release Wizard, or the Singe Exe File Name setting in the Releases view, as described in the following procedures.

Note • *The single executable file that you create accepts any command-line parameter that* **Setup.exe** *accepts.*

Task ***To package a build as a single executable file by using the Release Wizard:***

1. Launch the Release Wizard.
2. Navigate to the **General Options** panel.
3. Select the **Create a single file executable** check box.
4. By default, the file name **<project name>.exe** is entered in the **File Name** box. If you want the self-extracting executable file to have a different name, type a new file name in the box or select a path variable whose value defines the file name.
5. In the **Icon** box, you can specify the fully qualified name of the file that contains the icon that InstallShield should use when it creates the **Setup.exe** file at build time.

   To specify a file, type an explicit path or path variable, or click the browse button to open the **Change Icon** dialog box, in which you can click the **Browse** button to select a file.

   By default, the icon with index 0 is used; to specify a different icon, either select an icon in the **Change Icon** dialog box or append the icon's index or resource ID (preceded by a minus sign) to the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100.
6. If you want the executable file to extract the installation files to the target machine and not run the installation, enter **-extract_all:<path>** in the **Setup Command Line** box, where *<path>* is the desired target folder; for example:

   ```
   -extract_all:"C:/ProductName Setup Files"
   ```

7. Complete the **Release Wizard**; in the **Summary** panel, select the **Build the Release** check box and then click the **Finish** button.



Task ***To package a build as a single executable file by using the settings in the Releases view:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the release that you want to package as a single executable file.
3. Select the **Setup.exe** tab.
4. For the **Single .exe File Name** setting, type a file name or a path variable (enclosed in angle brackets—for example, **<MY EXE FILE NAME>**) whose value defines the file name.
5. In the **Setup.exe Icon File** setting, specify the fully qualified name of the file that contains the icon that InstallShield should use when it creates the **Setup.exe** file at build time.

   To specify a file, type an explicit path or path variable, or click the ellipsis button (**...**) to open the **Change Icon** dialog box, in which you can click the **Browse** button to select a file.

   By default, the icon with index 0 is used; to specify a different icon, either select an icon in the **Change Icon** dialog box or append the icon's index or resource ID (preceded by a minus sign) to the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100.
6. If you want the executable file to extract the installation files to the target machine and not run the installation, enter the following for the **Setup Command Line** setting:

   -extract_all:<path>

   where *<path>* is the desired target folder; for example:

   ```
   -extract_all:"C:/ProductName Setup Files"
   ```
7. Build the release.

To learn how to digitally signing your executable file, see Digital Signing and Security. For information about requiring end users to enter a password in order to launch the self-extracting executable file, see Password-Protecting Installations.

Leaving Files Uncompressed in a Disk Image



Project • *This information applies to InstallScript projects.*



Task ***To place some or all features' files uncompressed in the disk image, use the Media Layout panel in the Release Wizard:***

1. Launch the **Release Wizard** and navigate to the **Media Layout** panel.
2. Do either of the following:

- To place all features' files uncompressed in the disk image: Select the **CDROM Folder(s)** option. All features' files are placed in the disk image in the folders specified by the features' **CD-ROM Folder** properties. If no folder is specified for a feature, that feature's files are placed in the root of the disk image.
- To place some features' files uncompressed in the disk image:

  a. Select the **Custom** option, and then click **Next**. The **Custom Media Layout** panel opens.

  b. In the **Features in cabinets** box, select or clear the check boxes next to the features. If a feature's files are stored in cabinet files, clear the check box. If a feature's files should be placed in the disk image in the folder specified by the feature's **CD-ROM Folder** property, select the check box. If no folder is specified, that feature's files are placed in the root of the disk image.



***Tip** • If you want most of your features stored in cabinet files, click **Clear All** and then select the features to be placed uncompressed in the disk image. Conversely, if you want most of your features placed uncompressed in the disk image, click **Select All** and then clear the check boxes for the features to be stored in cabinet files.*

3. Complete the **Release Wizard**.



***Note** • Regardless of whether you selected Yes or No for the Compressed setting of a feature's component, by selecting the uncompressed (CD-ROM folder) option for the feature, its files will be uncompressed on the distribution media.*

Specifying the Required Execution Level for Your Setup Launcher on Windows Vista and Later Platforms



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

InstallShield lets you specify the minimum execution level required by your installation's **Setup.exe** file for running the installation (the setup launcher, any InstallShield prerequisites, the .msi file, and any Advanced UI or Suite/Advanced UI packages) on Windows Vista and later platforms. You can configure this for each individual release in your project.



Task ***To specify the required execution level for a release:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the release that you would like to configure.
3. Click the **Setup.exe** tab.
4. For the **Required Execution Level** setting, select the appropriate option.

The available options are:

- **Administrator**—**Setup.exe** requires administrative privileges to run. Administrators must provide consent, and non-administrators must provide credentials.
- **Highest available**—**Setup.exe** prefers administrative privileges. Administrators must provide consent to run it; non-administrators run it without administrative privileges. This is the default option for InstallScript and InstallScript MSI projects.
- **Invoker**—**Setup.exe** does not require administrative privileges, and all users can run it without administrative privileges. **Setup.exe** does not display any UAC messages prompting for credentials or for consent. This is the default option for Advanced UI, Basic MSI, and Suite/Advanced UI projects.

For Advanced UI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects, and for Basic MSI projects if the Setup Launcher setting is set to Yes, InstallShield embeds a Windows application manifest in the **Setup.exe** launcher as a resource. This manifest specifies the selected execution level. Operating systems earlier than Windows Vista ignore the required execution level. The execution level is defined in the manifest as follows:

```
<requestedExecutionLevel
  level="asInvoker"
  uiAccess="false"/>
```

Other valid values for the level attribute are **highestAvailable** and **requireAdministrator**.

If the Setup Launcher setting is set to No for a Basic MSI project, InstallShield does not embed the Windows application manifest in the **Setup.exe** launcher.

The benefit of elevating the required execution level in Basic MSI projects is that privileges can be elevated only once if necessary to run **Setup.exe**, and that these privileges can be carried over to all of the installation's InstallShield prerequisites and the Execute sequence of the .msi package without requiring multiple prompts for approval. Thus, if two of your InstallShield prerequisites require administrative privileges, for example, you can change this setting to Administrator, and then end users are prompted only once during the installation, before Windows Installer runs the **Setup.exe** file.

A similar benefit exists for Advanced UI and Suite/Advanced UI projects, where privileges can be elevated only once if necessary to run **Setup.exe**, and that these privileges can be carried over to all of the installation's Advanced UI or Suite/Advanced UI packages without requiring multiple prompts for approval.

Note, however, that if you elevate the privileges and also launch the application at the end of the installation, the elevated privileges are carried over to the application. In most cases, running an application with elevated privileges on Windows Vista and later platforms is discouraged.



Important • *InstallShield runs with elevated privileges. If you launch your installation from within InstallShield, those elevated privileges are carried over to your installation; thus, your installation automatically has elevated privileges. That may not reflect the behavior that end users will see if they are using Windows Vista or later. Therefore, if you are using Windows Vista or later on your development system, consider opening the release folder and launching the installation directly (instead of from within InstallShield).*

To quickly access your release folder so that you can launch your release directly, click the Open Release Folder on the Standard toolbar, or on the Tools menu, click Open Release Folder.

Note that an end user's installation experience is more secure when installations are run with only the permissions that they need. Unless an application is designed to be run only by system administrators, it should be run with the least privilege.

Specifying Whether a Product Should Be Advertised If Its InstallShield Prerequisites Are Run with Elevated Privileges



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Depending on how it is configured, an installation that includes InstallShield prerequisites may display a User Account Control (UAC) prompt for elevated privileges on Windows Vista and later systems at several different points during the installation:

1. When the end user launches the **Setup.exe** file
2. When the **Setup.exe** file launches a setup prerequisite that requires elevated privileges
3. When the **Setup.exe** file launches a feature prerequisite that requires elevated privileges
4. When the Windows Installer begins the Execute sequence of the .msi package

If you select Administrator for your installation's Required Execution Level setting in the Releases view, Windows Vista and later typically display only one UAC prompt; it is displayed when the end user launches the **Setup.exe** file. However, if you select Invoker for this setting, Windows Vista and later may display more than one UAC prompt during the installation. For example, Windows Vista and later may display a UAC prompt for a setup prerequisite that requires elevated privileges and another UAC prompt for the Execute sequence of the .msi package. If this scenario applies to your installation, you may want to specify that your installation should advertise and then run your .msi package to help end users avoid the UAC prompt for the .msi package. If this scenario does not apply to you, it is recommended that you avoid advertising the .msi package because it would not avoid a UAC prompt.



Tip *• The Require Administrative Privileges setting in the General Information view is where you specify whether the .msi package requires administrative privileges. The Behavior tab in the InstallShield Prerequisite Editor is where you specify whether a prerequisite requires administrative privileges. For more information about other InstallShield settings that may affect whether Windows Vista and later displays UAC prompts, see* *Minimizing the Number of User Account Control Prompts During Installation**.*



Task ***To specify whether your .msi package should be advertised:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the release that you would like to configure.
3. Click the **Setup.exe** tab.
4. For the **Advertise If Prerequisites Are Elevated** setting, select the appropriate option.

The available options are:

- **Advertise: Silent**—Indicates that if setup prerequisites in the installation are successfully installed with elevated privileges, the .msi package should be advertised and run silently (without a user interface). Then the Execute sequence of the main installation does not require an additional UAC prompt for consent or credentials.

- **Advertise: Full UI**—Indicates that if setup prerequisites in the installation are successfully installed with elevated privileges, the .msi package should be advertised and run with a full user interface. Then the Execute sequence of the main installation does not require an additional UAC prompt.
- **No**—Indicates that the .msi package should not be advertised. When end users run the installation, one or more UAC prompts may be displayed to install the setup prerequisites. If the Execute sequence of the .msi package also requires elevation, an additional UAC prompt may be displayed before the Windows Installer begins the Execute sequence.



***Important** • The package must support advertisement in order for either of the advertise options to work. Advertisement is not instantaneous, and it adds extra delays to the installation. In addition, unexpected behavior may occur if the end user clicks Cancel after advertisement but before the main part of the installation has finished. For example, advertised shortcuts for your product may appear on the desktop before the main installation begins, and a confused user canceling the main installation may leave your package advertised but not fully installed. Therefore, in some cases, it may be better to leave this setting as No to allow the extra UAC prompt and avoid product advertisement.*

A common goal is for an installation to display only one UAC prompt. The advertise options for the Advertise If Prerequisites Are Elevated setting facilitate this but do not guarantee it in all situations. For example, any time that an installation causes a restart, the installation process returns to limited privileges after the restart. The subsequent privilege elevation may display an additional UAC prompt, whether the elevation is required for a setup prerequisite, for a feature prerequisite, or for the .msi package. When the restart comes between the last prerequisite and the .msi package, the .msi package is not advertised. Following are some examples of different outcomes that may occur when you select one of the advertise options for this setting:

- The prerequisites require elevation and install successfully on the target machine. The product is advertised and installed. The product installation does not display a UAC prompt.
- One of the prerequisites in the installation may require a restart. After the restart for the prerequisite occurs, either no additional prerequisites are installed or none of the other prerequisites that are installed require elevation. Since the last prerequisite that is installed in this scenario is not a simple success of an elevated prerequisite, advertisement of the .msi package does not occur.
- None of the prerequisites in the installation need to be installed because they are already present on the target machine; therefore, advertisement of the .msi package does not occur.
- None of the prerequisites in the installation require elevation; therefore, advertisement of the .msi package does not occur.
- Elevated prerequisites are successfully installed. However, for this situation, the package is a minor upgrade for a product that is already installed. That is, the product code in the package matches the product code of the product that is already present on the target machine. Advertisement of the .msi package does not occur. UAC prompts may or may not be displayed; it depends on whether a suitable digital certificate is included in the earlier installation and in the patch.

Password-Protecting Installations

For added security, you can password-protect your installation package. When you password-protect your installation, any end user who wants to install your package must enter a case-sensitive password to open your installation.

You can activate password protection in the Password & Copyright panel of the Release Wizard.

Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

InstallShield lets you specify the run-time location of the InstallShield prerequisites that are included with your installation.

Task ***To specify where all of the InstallShield prerequisites should be located for a release, do one of the following:***

- Use the **InstallShield Prerequisites** panel of the **Release Wizard**.
- In the **Releases** view, select the release. Then on the **Setup.exe** tab, for the **InstallShield Prerequisites Location** setting, select the appropriate option.

Task ***To specify different locations for each InstallShield prerequisite:***

1. In the **Redistributables** view (in Basic MSI and InstallScript MSI projects) or the **Prerequisites** view (in InstallScript projects), specify the appropriate location for each InstallShield prerequisite. For more information, see Specifying a Run-Time Location for a Specific InstallShield Prerequisite.
2. In the View List under **Media**, click **Releases**.
3. Select the release that you want to build.
4. Click the **Setup.exe** tab.
5. For the **InstallShield Prerequisites Location** setting, select **Follow Individual Selections**.

Tip *• As an alternative, you can select the Follow Individual Selections option in the InstallShield Prerequisites panel of the Release Wizard.*

Note that if an InstallShield prerequisite is added to a project as a dependency of another prerequisite, the location for the prerequisite dependency follows the location setting of the prerequisite that requires it.

If you build a release that includes InstallShield prerequisites and both of the following are true, one or more build errors may be generated:

- You specify for the InstallShield prerequisites location that the prerequisites should be extracted from **Setup.exe** or copied from the source media (instead of being downloaded from the Web to the end user's computer).
- The prerequisite files are not on your computer.

To eliminate the build errors, remove the InstallShield prerequisite from your project, download the InstallShield prerequisite from the Internet to your computer, or change the InstallShield prerequisites location for the release to the download option; then rebuild the release.

Specifying the Run-Time Location for Advanced UI or Suite/Advanced UI Packages at the Release Level



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you specify the run-time location of the packages that are included with your Advanced UI or Suite/Advanced UI installation.



Tip *• If you are configuring an Advanced UI or Suite/Advanced UI project for an update setup launcher, the run-time location of the packages must be either extracted from the setup launcher or downloaded from the Web. The update setup launcher cannot rely on packages that are stored on the source media.*

For more information, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.



Task ***To specify where all of the packages should be located for a release:***

1. In the View List under **Media**, click **Releases**.
2. Select the release that you want to build.
3. Click the **Setup.exe** tab.
4. In the **Package Location** setting, select the appropriate option.



Task ***To specify different locations for each package:***

1. In the View List under **Organization**, click **Packages**.
2. Specify the appropriate location for each InstallShield prerequisite. For more information, see Specifying a Run-Time Location for a Specific Package in an Advanced UI or Suite/Advanced UI Project.
3. In the View List under **Media**, click **Releases**.
4. Select the release that you want to build.
5. Click the **Setup.exe** tab.
6. In the **Package Location** setting, select **Follow Individual Selections**.

Note that if a package is added to a project as a dependency of another package, the location for the package dependency follows the location setting of the package that requires it.

If you build a release that includes package and both of the following are true, one or more build errors may be generated:

- You specify for the package location that the packages should be extracted from **Setup.exe** or copied from the source media (instead of being downloaded from the Web to the end user's computer).
- The package files are not on your computer.

To eliminate the build errors, remove the package from your project, add the package to your computer, or change the package location for the release to the download option; then rebuild the release.

Digital Signing and Security



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

You can digitally sign your installation and your application files to assure end users that neither your installation nor the code within your application has been tampered with or altered since publication.

The Signing tab is where you specify the digital signature information—including the digital certificate files that a certification authority grated to you—that InstallShield should use to sign your files.

The Signing tab is also where you specify which files in your installation should be digitally signed by InstallShield at build time. InstallShield enables you to sign any and all of the following files in a release, depending on what type of project you are using:

- Windows Installer package (.msi file) for Basic MSI and InstallScript MSI projects
- Merge module package (.msm file) for Merge Module projects
- **Setup.exe** file for Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects
- Media header file for InstallScript projects
- Any files in your release, including your application files



***Windows Logo** • All executable files (including .exe, .dll, .ocx, .sys, .cpl, .drv, and .scr files) in an installation must be digitally signed for the Windows logo program.*

To learn more about the various settings on the Signing tab, see Signing Tab for a Release.

Certification Authorities

A certification authority is an organization such as VeriSign that issues and manages digital certificates (also known as digital IDs). The certification authority validates the requester's identity according to prescribed criteria and issues a digital certificate. Obtaining a digital certificate requires providing the certificate authority with specific information about your company and your product.

For a list of certification authorities, see the Windows Root Certificate Program member list on the MSDN Web site.

SHA-1 vs. SHA-256 Certificates

InstallShield enables you to use digital certificates that use the SHA-256 or SHA-1 hashing algorithm for signing your installations and files at build time.

SHA-256 is favored over SHA-1, which is being deprecated because of the potential for security vulnerabilities. Microsoft announced that Windows will stop trusting items that were signed and timestamped with SHA-1 certificates after January 1, 2016. In addition, certification authorities—the organizations that issue certificates—are phasing out the creation of SHA-1 certificates. Thus, it is recommended that you replace any SHA-1 certificates in your InstallShield projects with SHA-256 certificates. For the latest information and more specific details, check with your certification authority.

If your project is configured to sign with a SHA-256 certificate, InstallShield uses a SHA-256 hash in the signature of the files that it signs at build time. If your project is configured to sign with a SHA-1 certificate, InstallShield uses a SHA-1 hash. Note that use of a SHA-1 certificate triggers build warning -7346 to alert you about the SHA-1 usage.

Beginning with InstallShield 2016, support for SHA-256 digital certificates for Windows Installer and InstallScript projects was enhanced to:

- Give you the ability to specify a digest type
- Allow you to specify a SHA-256 timestamp server in **settings.xml**
- Handle similarly-named certificates in the Certificate Store



Important • *Any new signatures created or timestamped after Jan 1, 2016 must be SHA-256-based signatures. Any files signed with an SHA-1 certificate need to have a timestamp showing a date and time prior to Jan 1, 2016 in order to continue to be supported. Those files will still be allowed through the 'Mark-of-the-web" system until Jan 14, 2020, when all SHA-1 support will stop in all current versions of Windows.*

Using Certificate Files or Certificates in Certificate Stores to Generate a Digital Signature

When you are specifying the digital signature information that you want to use for signing your files and installations, InstallShield lets you choose between the following options:

- You can specify the .pfx certificate file on your machine that you want to use.
- You can reference a certificate store that contains the certificate that you want to use.

Option 1—.pfx File

You can use a personal information exchange file (.pfx) to digitally sign your installation and your application. The following tools enable you to create a .pfx file from a .pvk file and .spc file:

- **PVK2PFX.exe**—This is part of the Windows Platform SDK, and it is also included with Microsoft Visual Studio 2005.
- **pvkimprt.exe**—You can download this PVK Digital Certificate Files Importer tool from the downloads area on the Microsoft Web site (http://www.microsoft.com/downloads/details.aspx?FamilyID=F9992C94-B129-46BC-B240-414BDFF679A7&displaylang=EN).

The .pfx file is typically associated with a password.

Option 2—A Certificate in a Certificate Store

If you store your digital certificate in a certificate store, you can reference in your project the certificate store that contains the certificate that you want to use. If you use this method for signing, you will need to specify information such as the store name (Personal, Trusted Root Certification Authorities, Enterprise Trust, Intermediate Certification Authorities), the store location (user or machine), and the subject that identifies the specific certificate that you want to use.

If you configure your project to use a certificate that was imported with password protection into a store, Windows prompts for the password at build time when InstallShield is attempting to sign your project's files. The strong key protection that Windows uses does not permit InstallShield to provide the password to the cryptographic provider.

Adding a Timestamp to Digital Signatures

When you specify digital signature information for a release, InstallShield timestamps the digital signature at build time by default. A digital certificate's timestamp indicates the day and time when a file was signed; it also helps to prove that the certificate was not expired at the moment of signing. A timestamp from a trusted timestamp server typically extends the life of the digital signature beyond the certificate's validity period, protecting against certificate expiration.

To learn how to change the default timestamp server that InstallShield uses, or for information on how to disable timestamping, see Changing the Timestamp Server for Digital Signatures.

Digitally Signing a Release and Its Files at Build Time

InstallShield lets you configure digital signing settings for a release. At build time, InstallShield uses the settings that you have configured to sign your installation package, your **Setup.exe** file, and any other files in your release that meet the criteria that you have defined.

Task ***To configure digital signing for your release and its files:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the release that you want to sign.
3. Click the **Signing** tab.
4. Configure the following settings as appropriate:
   - **Certificate URL**
   - **Digital Certificate File**—Click the ellipsis button (**...**) in this setting. The Certificate Selection dialog box opens, enabling you to specify either the location of the .pfx file or information about the certificate store that contains the certificate.

- **Certificate Password**—Note that if you configure your project to use a certificate that was imported with password protection into a store, Windows prompts for the password at build time when InstallShield is attempting to sign your project's files. The strong key protection that Windows uses does not permit InstallShield to provide the password to the cryptographic provider.
- **Signature Description**

5. In the **Sign Output Files** setting, specify which files (**Setup.exe**, the .msi package, both of those files, or neither of those files) you want to be signed.
6. In the **Sign Files in Package** setting, specify whether you want to sign additional files in your installation.

   If you select **Yes**, use the other settings under the **Sign Files in Package** setting to indicate which files and file patterns should be signed and which should not be signed.

   Note that the files and file patterns that should not be signed override any files and file patterns that should be signed. For example, if you specify ***.exe** in an **Include** setting and in an **Exclude** setting, InstallShield does not sign any .exe files.



***Tip** • For detailed information about any of the settings on the Signing tab, see Signing Tab for a Release.*

At build time, InstallShield signs the files as specified on the Signing tab. If the release is for an installation that includes merge modules, note that the files are signed before the merge module is merged.

Digitally Signing a Release After It Has Been Built From the Command Line



***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

As an alternative to having InstallShield sign your application at build time, you can use the iSign application (**iSign.exe**) to digitally sign a release of an InstallScript project from the command line after you have built the release.

iSign.exe is in the following directory:

*`InstallShield Program Files Folder`***\System**

For syntax and available command-line parameters for **iSign.exe**, see iSign.exe.

Release Flags



***Project** • The following project types include support for release flags:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

When you build your release, you can include and exclude features, InstallShield prerequisites, and chained .msi packages depending on the type of release. For example, if you are creating a trial version of your product and do not want to include all the features in the build, you can flag features and then specify those flagged features under the product configuration or the release.

Task ***To filter features, InstallShield prerequisites, and chained .msi packages based on release flags:***

1. Assign release flags to the features, InstallShield prerequisites, and chained .msi packages.

   For detailed information on how, see Assigning Release Flags to Features, Assigning Release Flags to InstallShield Prerequisites, or Assigning Release Flags to a Chained .msi Package.

2. Specify which flags to include in the release if appropriate.

The scenario of a trial version versus a full version is a good example of why you might use release flags. The table below gives an example of four different features that may be available in a product, depending on which version the end user receives.

In this example, the full version comes with the application's executable file, help files, a spellchecker, and an add-on pack that is included as a chained .msi package. The trial version does not include the add-on pack, but it does include an upgrade pack. Rather than having to create two separate installation projects for what is essentially one product, you can flag those features that are specific to certain versions. For example, the add-on feature is flagged as **Full** and the upgrade feature as **Trial**.

Table 4-7 • Sample Release Flags

Feature, InstallShield Prerequisite, or Chained .msi Package	Version	Release Flag
Windows Installer 4.5 Redistributable (InstallShield prerequisite)	All Versions	Not applicable
Program_Executable (feature)	All Versions	Not applicable
Help_Files (feature)	All Versions	Not applicable
Spellchecker (feature)	Full Version	Full
Add_Ons (chained .msi package)	Full Version	Full
Upgrade_Pack (feature)	Trial Version	Trial

If you build your release through the Release Wizard, you are given the option of including any flagged features, flagged InstallShield prerequisites, and flagged chained .msi packages into your release. By default, the release contains all features, InstallShield prerequisites, and chained .msi packages. If you specify the release flag **Full** in the Filtering Settings panel of the Release Wizard, only the **Spellchecker** feature and the **Add_Ons** chained .msi package—along with all features and InstallShield prerequisites without release flags—are built into your release. The upgrade pack is not included in the installation.



Tip • *By default, all features, InstallShield prerequisites, and chained .msi packages are included in a release. When you specify a release flag in either the Releases view or the Release Wizard, only the unflagged features, the unflagged InstallShield prerequisites, the unflagged chained .msi packages, the features that contain the specified flag, the InstallShield prerequisites that contain the specified flag, and the chained .msi packages that contain the specified flag are included in your installation. InstallShield does not include in the release any features, InstallShield prerequisites, or chained .msi packages that have a release flag that you did not include in the Releases view or Release Wizard.*

Product Configuration Flags vs. Release Flags



Project • *The following project types include support for product configuration flags and release flags:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

You can specify which flags to include in your installation both on the release, known as *release flags*, and on the product configuration, known as *product configuration flags*. Although release flags are exposed both as a setting of the release and in the Release Wizard's Filtering Settings panel, you can edit the product configuration flags only on the General tab of the product configuration in the Releases view.

Product configuration flags complement release flags. That is, any feature flags that you specify for the product are combined with the flags on the individual release that you are building, and all of the features, InstallShield prerequisites, and chained .msi packages with the specified flags are built into the release. Be careful when specifying flags in the Release Wizard, because you cannot see which flags are included at the product configuration level.

To demonstrate how product configuration and release flags interact, consider a project with the following features, InstallShield prerequisites, and chained .msi packages.

Table 4-8 • Sample Features, InstallShield Prerequisites, and Chained .msi Packages

Feature, InstallShield Prerequisite, or Chained .msi Package	Version	Release Flag
Windows Installer 4.5 Redistributable (InstallShield prerequisite)	All Versions	Not applicable
Program_Executable (feature)	All Versions	Not applicable
Help_Files (feature)	All Versions	Not applicable
Spellchecker (feature)	Full Version	Full
Add_Ons (chained .msi package)	Full Version	Full
Upgrade_Pack (feature)	Trial Version	Trial

The following table explains which features, InstallShield prerequisites, and chained .msi packages are included in your installation based on the combination of product configuration flags and release flags:

Table 4-9 • Combining Product Configuration Flags and Release Flags

Product Configuration Flags	Release Flags	Features, InstallShield Prerequisites, and Chained .msi Packages in Installation
<None>	<None>	Windows Installer 4.5 Redistributable, Program_Executable, Help_Files, Spellchecker, Add_Ons, Upgrade_Pack
<None>	Full	Windows Installer 4.5 Redistributable, Program_Executable, Help_Files, Spellchecker, Add_Ons
Trial	<None>	Windows Installer 4.5 Redistributable, Program_Executable, Help_Files, Upgrade_Pack
Full	Trial	Windows Installer 4.5 Redistributable, Program_Executable, Help_Files, Spellchecker, Add_Ons, Upgrade_Pack

Filtering Based on Release Flags



Project • *The following project types include support for product configuration flags and release flags:*

- *Basic MSI*
- *InstallScript MSI*
- *Suite/Advanced UI*

Once you have assigned release flags to your features and InstallShield prerequisites, you can create a release that includes the ones that are based on those flags. By default, all features, InstallShield prerequisites, and chained .msi packages are included in a release. Once you specify a flag in either the Releases view or the Release Wizard, only the unflagged features, the unflagged InstallShield prerequisites, the unflagged chained .msi packages, the features that contain the specified flag, the InstallShield prerequisites that contain the specified flag, and the chained .msi packages that contain the specified flag are included in your installation. To include only unflagged features, unflagged InstallShield prerequisites, and unflagged chained .msi packages, specify a flag that does not exist. For example, you might use **NoFlags**. This way, only unflagged features, unflagged InstallShield prerequisites, and unflagged chained .msi packages are built into a release.

It is possible to include a subfeature but not its parent feature. In such a case, the subfeature is built into the release as a top-level feature, and its parent is excluded from the release.

You can filter features, InstallShield prerequisites, and chained .msi packages in the Release Wizard or in the Releases view. Both methods are described below.

Filtering in the Release Wizard

The easiest way to create a release is to use the Release Wizard. The Filtering Settings panel of this wizard enables you to specify which flags you want to include in your release.

Filtering in the Releases View

You can also use the Releases view to specify flags to include. You can include flags both at the product configuration level and on the release itself. For more information, see Product Configuration Flags vs. Release Flags.



Task ***To specify product configuration flags:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the product configuration that you want to modify. Its settings are displayed on the **General** tab in the right pane.
3. For the **Product Configuration Flags** setting, enter the flags that you would like to include. To include more than one flag, separate each with a comma.



Task ***To specify release flags:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the release that you want to modify. Its settings are displayed on tabs in the right pane.
3. For the **Release Flags** setting, enter the flags that you would like to include in this release. To include more than one flag, separate each with a comma.

Using One Project to Create Installations for Multiple Product Configurations



Project • *This information applies to InstallScript projects.*

You may want to release your product with more than one configuration—for example, an evaluation release and a full release. Within a single project, you can build releases containing different subsets of the project's features; with a single installation script using preprocessor directives, you can build releases with different run-time behaviors.



Task ***To determine a release's included features at build time:***

1. Launch the **Release Wizard** and navigate to the **Features** panel.
2. Select the **Specify the features to be included below** option.
3. In the **Features** explorer, select the check boxes of the features that you want to include in the product configuration, and clear the check boxes of the features that you do not want to include.

4. Complete the other panels in the **Release Wizard**.



Task ***To determine a release's run-time behavior using preprocessor directives:***

1. In your script, use preprocessor directives to execute different code for different product configurations; for example:

```
#ifdef EVAL_RELEASE
    /* Evaluation-specific code */
#elif FULL_RELEASE
    /* Full release-specific code */
#endif
```

2. In the **Compiler Preprocessor Defines** box (on the **General Options** panel of the **Release Wizard**) or the **Compiler Preprocessor Defines** setting (on the **Build** tab in the **Releases** view), specify the preprocessor constant that corresponds to the code that you want the release to execute.
3. Build the release.

Typical Web Installation vs. One-Click Install

When downloading an application from the Internet, many end users experience a long and confusing series of instructions explaining how to download and run the application properly. The typical download can take as many as 10 steps. It can be a frustrating and time-consuming process that is not always successful. End users who fail to download the application properly must rely on product support to assist them, or they give up completely and never gain access to the application. This results in time lost for both the consumer and the development team.

With a One-Click Install setup, however, the end user can download an application with minimum effort and can begin using the application immediately. Compare the installation processes for both typical and One-Click Install setups below.

Typical Internet Installation Process

A typical Internet installation might happen like this:

1. The end user navigates to the Web page and clicks the link to download the application.
2. A dialog opens. The dialog prompts the end user to agree to a software licensing agreement.
3. The end user is given the opportunity to save the installation's executable file to disk. The end user must remember this location to access the installation's executable file later.
4. The file transfer begins.
5. The end user must search for the installation executable file and then launch it.
6. The installation executable file unpacks and executes.
7. If any other resources are necessary for the proper execution of the application, the end user must download them at this time.
8. The installation executes and begins to walk the end user through the installation process.
9. The end user makes several choices, such as the disk location of the application executable file.
10. The application file transfer begins.

11. The end user locates the application executable file on disk and launches the application.

One-Click Install Installation Process

With a One-Click Install setup, installations are easier and faster for the end user. A One-Click Install setup can happen like this:

1. The end user navigates to the Web page and clicks the download button.
2. A dialog opens. The dialog shows the progress of the file transfer. Once the installation has been downloaded, the installation runs.
3. The end user begins using the application.

One-Click Install setups keep the end user on your Web page, instead of searching for an executable file to run. One-Click Install setups make it possible for the end user to concentrate on using the application, not struggling to download it.

One-Click Install Installations in InstallScript Projects



Project • *This information applies to InstallScript projects.*

A One-Click Install installation includes a setup player (**Setup.ocx**) that downloads and then launches the **Setup.exe** file with the appropriate command line. This enables end users who are using Windows Vista and later with limited privileges to run the installation; if elevated privileges are required because of the required execution level specified in the installation's manifest, the appropriate User Account Control (UAC) prompt is displayed when the **Setup.exe** file is launched. If end users have an earlier version of Windows on their systems, the manifest is ignored, but the other behavior is the same.

The **Setup.exe** file for One-Click Install installations does not include any COM information.

Like any other installation, an Internet installation can use features such as updating, maintenance mode, multi-instance installation, silent installation—in short, any feature that is available for any InstallShield installation.

The following InstallScript functions are Internet-enabled:

Table 4-10 • Internet-Enabled InstallScript Functions

Function	Description
CopyFile	Copies a file that is specified by a valid URL, or transmits a CGI or ASP request.
ExistsDir	Checks the existence of a folder that is specified by a valid URL.
GetFileInfo	Gets information about a file that is specified by a valid URL.
GetLine	Reads lines from a file that is specified by a valid URL.

Table 4-10 • Internet-Enabled InstallScript Functions (cont.)

Function	Description
Is	Checks the following: • Whether the installation is being run from the Internet • The existence of a file or folder that is specified by a valid URL • Whether a given string is a URL • The validity of a URL
OpenFile	Opens a file that is specified by a valid URL.
ParseUrl	Retrieves the parts of the specified URL.
ReadBytes	Reads bytes from a file that is specified by a valid URL.
SeekBytes	Repositions the file pointer within a file that is specified by a valid URL.

One-Click Install Objects



Project • *This information applies to InstallScript projects.*

The objects within One-Click Install's setup player are Player and Ether. The Player object is the highest level of the object model, followed by Ether.

Player Object

Before you can use the setup player, you must create an instance of it. For more information, see Using the Setup Player.

The Player has one method and one property.

- Open (<URL>)—Specifies the location of the data .cab files and returns a reference to an Ether object. The argument consists of a URL representing the path to a web server, a UNC path, or the path to the location of the files on your local drive.
- LastError—Integer value representing an error code thrown by a method, usually Open().

Examples

```
Player.Open("http://www.mydomain.com/myproduct");
Player.Open("file://\\server\myproduct");
Player.Open("file://c:/my installations/myproduct/media/default");
```

LastError Method



Project • *This information applies to InstallScript projects.*

This is the LastError method of the Player object. This is an integer value containing the error code for an error thrown by a method of the Player, usually Open().

Syntax

```
var nError = player.LastError;
```

Example

```
var ether = Player.Open("http://www.installshield.com")
```

Parameters

None.

Return Values

Table 4-11 • Return Values for the LastError Method

Return Code	Description
-2147186688	No permissions were granted. Clicking Deny on the Java Security dialog throws this error.
-2147186686	Failed to update or install the Setup Player ActiveX control. Verify your ability to download from the URL specified in the Codebase attribute of the APPLET tag.
-2147186685	Failed to instantiate ISPlayer or initialization of the ISPlayer failed.
-2147186684	Failed to call player.Open.
-2147186683	Player.Open call timed out waiting for applet initialization.
-2147166892	Requested URL not found. Verify that your installation files are available at the URL specified during the player.Open call.
-2147166895	Unable to access protected resource. This error is returned when attempting to access an installation that requires authentication and the end user does not enter the correct information.
-2147012889	The server name could not be resolved. Verify that the URL used in your player.Open call. Misspelling the domain name the URL is a common cause of this error.



Note • *For the first six errors listed, the end user can check the Java console for detailed error information.*

Open Method



Project • *This information applies to InstallScript projects.*

The Open method of the Player object accepts one string argument that specifies the URL of the installation's cabinet files.

Syntax

```
var ether = Player.Open("<URL string>")
```

Example

```
var ether = Player.Open("http://www.installshield.com")
```

Parameters

Table 4-12 • Valid Parameter for the Open Method

Parameter	Description
<URL string>	Specifies the URL of the installation's cabinet files.

Return Values

The Open method returns a reference to an Ether object.

Ether Object

The Ether object supports the following methods:

Table 4-13 • Methods Supported by the Ether Object

Method	Description
GetLastError	Returns a numeric value indicating the result of the installation.
IsPlaying	Returns a Boolean value indicating if the installation is running.
Play	Starts the installation.
SetProperty	Sets a property to a certain value.

GetLastError Method

After calling the Ether object's Play method, call the GetLastError method to find out the result of the installation. GetLastError tells you if an error occurred, if the installation finished successfully, or if the installation was canceled.

Syntax

```
ether.GetLastError( );
```

Example

It is best to poll GetLastError as in the following example:

```
var intervalID = 0;

function startInstall() {
    if (ether)
    {
        ether.Play();
        if (intervalID == 0)
            intervalID = window.setInterval("IsSetupFinished()", 3000);
    }
}

function IsSetupFinished()
{
    var nResult = ether.GetLastError();
    if (nResult != 0)
    {
        if (nResult == SETUP_FINISHED) // setup has completed successfully
            /* to do */ ;
        if (nResult == SETUP_ERR_CANCELLED) // setup was cancelled
            /* to do */ ;

        clearInterval(intervalID);
        intervalID = 0;
    }
}
```

Parameters

None.

Return Values

A complete list of GetLastError return values is in **Disk1_graphics\Utilities.js** under the release's **Disk Images** folder, and is given below.

- SETUP_RUNNING
- SETUP_FINISHED
- SETUP_ERR_GENERAL
- SETUP_ERR_LOADMEDIA
- SETUP_ERR_INSTALLKERNEL
- SETUP_ERR_STARTKERNEL
- SETUP_ERR_OPENCAB
- SETUP_ERR_INSTALLSUPPORT
- SETUP_ERR_SETTEXTSUB
- SETUP_ERR_INITINFO
- SETUP_ERR_GETSETUPDRIVER
- SETUP_ERR_INITPROPERTIES
- SETUP_ERR_RUNINSTALL

- SETUP_ERR_UNINSTALLSUPPORT
- SETUP_ERR_EXTRACTBOOT
- SETUP_ERR_DOWNLOADFILE
- SETUP_ERR_CANCELLED

IsPlaying Method

The IsPlaying method of the Ether object returns a Boolean value indicating whether the installation is running.

Syntax

```
var <variable> = ether.IsPlaying();
```

Example

```
var bPlaying = ether.IsPlaying();
```

Parameters

None.

Return Values

Table 4-14 • Return Values for the IsPlaying Method

Return Value	Description
true	Indicates that the installation is running.
false	Indicates that the installation is not running.

Play Method

The Play method of the Ether object starts the installation.

To see an example of the use of this method, build a release using the Create a default Web page option in the Release Wizard's Internet Options panel or the Release property grid's Create Default Web Page property, then examine the code in the generated Web page (**Setup.htm**).

Syntax

```
ether.Play();
```

Parameters

None.

Return Values

None.

SetProperty Method

The SetProperty method of the Ether object accepts two arguments. The first, a string, represents the name of the property is::CmdLine. The second is a string specifying **Setup.exe** command-line parameters.

Syntax

```
ether.SetProperty("is::CmdLine","Setup.exe command-line options");
```

Example

```
ether.SetProperty("is::CmdLine","-l0x0407");
```

Parameters

The following table contains the parameters for the SetProperty method.

Table 4-15 • Parameters for the SetProperty Method

Property Name	Property Value
is::CmdLine	A string specifying **Setup.exe** command-line parameters. For a list of supported parameters, see Setup.exe and Update.exe Command-Line Parameters.

Return Values

None.

Digital Signatures for One-Click Install Installations

When an end user runs a One-Click Install installation, the setup player (**Setup.ocx**) needs to verify the digital signatures for following files:

- **Setup.ocx**
- **Setup.exe**
- **Data1.hdr**
- **ISSetup.dll**

If any of those files are unsigned or the signature is corrupted, the setup player displays a single prompt to inform the end user that the signature could not be verified. If all files are properly signed and your certificate is not yet always trusted, the setup player displays a prompt to ask the end user to authorize it (similar to the one that is displayed when you run an executable file that you downloaded).

InstallShield signs the **Setup.ocx** file for you. The certificate that is used to sign the **Setup.exe**, the **Data1.hdr**, and **ISSetup.dll** files must match. This happens automatically if you select the options for signing the **Setup.exe** file and the media header.

Replacing Obsolete Properties and Methods

InstallShield's simplified model for Internet installations eliminates several methods and properties that the Ether object and its subobjects had in InstallShield Professional 6.x. The following is a list of those methods and properties, and how to replace them if you are converting an InstallShield Professional 6.x installation Web page to work with an InstallShield installation.

AskDestPath

To display the equivalent dialog during the installation, call the InstallScript function **AskDestPath** in your installation script. To pass a destination path to the installation from the Web page, set a script-defined property.

CommandLine

Replace a line like the following:

```
ether.CommandLine = "<string value>";
```

with a line like the following:

```
ether.SetProperty("is::CmdLine","<string value>");Features, SetState
```

To offer the equivalent functionality during the installation, call the InstallScript function **SdFeatureTree** in your installation script. To pass a feature selection to the installation from the Web page, set a script-defined property; in addition, in the installation script, use its value to conditionally call **FeatureSelectItem**.

Features, GetState

To get a feature's selection state during the installation, call the InstallScript function **FeatureIsItemSelected** in your installation script. A Web page cannot get a feature's selection state.

FileToOpen

To open a file during the installation, call the InstallScript function **LaunchApplication** in your installation script. To pass a file name to the installation from the Web page, set a script-defined property.

GetDownloadSize, FileLevelCosting

To get the required disk space during the installation, call the InstallScript function **FeatureGetTotalCost** in your installation script. (Feature dialogs such as **SdFeatureTree** display the required disk space.) A Web page cannot get the required disk space.

Language

Replace a line like the following:

```
ether.Language = "<language code>";
```

with a line like the following:

```
ether.SetProperty("is::CmdLine","-l<language code>");LegacyMode
```

By default, Internet installations run in *legacy mode*—that is, with the installation's user-interface events generated as appropriate. There are two ways to duplicate the effect of setting ether.LegacyMode=false; on your Web page:

- Run the installation silently with the following line:

```
ether.SetProperty("is::CmdLine","-s");
```

- Set a script-defined property or check the return value of Is(WEB_BASED_SETUP), and conditionally call the installation's user-interface event handler functions in the OnShowUI handler.

ProductName

A Web page cannot get this information.

GetTargetDir

A Web page cannot get this information.

GetTextSub

A Web page cannot get this information.

IsInstalled

A Web page cannot get this information.

SetSetupType

To specify a setup type during the installation, call the InstallScript function **FeatureSetupTypeSet** in your installation script. To display a setup type selection dialog during the installation, call **SdSetupType** in your installation script. To pass a setup type to the installation from the Web page, set a script-defined property.

SetTargetDir

To specify a target directory during the installation, assign a value to the system variable TARGETDIR in your installation script. To pass the target directory to the installation from the Web page, set a script-defined property.

SetTextSub

To specify a text substitution during the installation, assign a value to the TextSub object's Value property in your installation script. To pass a text substitution to the installation from the Web page, set a script-defined property.

Default Web Page (Setup.htm)



Project • *This information applies to InstallScript projects.*

A default Web page called **Setup.htm** is created in your disk image folder when you build your One-Click Install installation using one of the following methods:

- Select the **Create a default Web page for the setup** check box on the Internet Options panel in the Release Wizard.
- Select Yes for the Create Default Web Page setting on the Internet tab in the Releases view.

You can use this page as it is, customize it however you want, or refer to it as an example of using the Setup Player.

Note that the aforementioned Web page creation settings are disabled if the Generate One-Click Install setting in the Releases view is set to No.

System Requirements for One-Click Install Installations



Project • *This information applies to InstallScript projects.*

You must have a Web server that supports HTTP 1.1 on the hosting system. Note that FTP does not support One-Click Install installations.

Alternate Ways to Call the Setup Player



Project • *This information applies to InstallScript projects.*

It is not necessary to call the Setup Player from a Web page. It is possible to call the Player from other sources such as a login script or a Visual Basic or Visual C/C++ application. Any programming tool that supports ActiveX controls can call the Player.

Sample Visual Basic Script for Calling the Player

```
Dim player, ether
Set player = CreateObject("Setup.Player.Automation Interface Version")
Set ether = player.open("http://www.mydomain.com/mysetup")

If ether.IsPlaying() Then
    MsgBox "Setup is running."
Else
    MsgBox "Setup is not running."
End If

Set player = Nothing
Set ether = Nothing
```

HTTPS and One-Click Install Installations



Project • *This information applies to InstallScript projects.*

Your One-Click Install installation can be run under the secure hypertext transfer protocol (HTTPS)—that is, HTTP using the Secure Sockets Layer (SLL). The only modification that your installation requires in order to run with HTTPS is to change http:// to https:// in URLs that are passed to Internet-enabled functions.

Creating One-Click Install Installations

You can create a One-Click Install installation using the Release Wizard.



Task ***To create a One-Click Install installation for an InstallScript project:***

1. Launch the **Release Wizard**.
2. Navigate to the **Internet Options** panel.
3. Select the **Generate One-Click Install** option.
4. Select the **Create a default Web page for the setup** check box or specify the location of a custom Web page in the **Enter the URL to launch** box.
5. Specify other options as desired.
6. Complete the rest of the Release Wizard panels.
7. Build your release.



Task ***To create a One-Click Install installation for a Basic MSI or InstallScript MSI project:***

1. Launch the **Release Wizard**.
2. Navigate to the **Media Type** panel.
3. In the **Media Type** list, click **Web**.
4. Navigate to the **One Click Install** panel, and select the **Generate One-Click Install** option.
5. Specify the file name for your .htm or .cab file.
6. Indicate the browsers that you want your One-Click Install installation to support.
7. Complete the rest of the Release Wizard panels.
8. Build your release.

Using the Setup Player



Project • *This information applies to InstallScript projects.*

The Setup Player is a **Setup.ocx** file that downloads and then launches the **Setup.exe** file with the appropriate command line. The **Setup.ocx** file also supports launching a setup executable file with a name other than **Setup.exe**; for this to work correctly, the [Startup] section of the **Setup.ini** file should contain the following key:

`LauncherName=`*`SetupLauncher`*

where *`SetupLauncher`* is the file name of the alternate setup launcher.

For an example of using the Setup Player in a Web page, build a release using the Create a default Web page option in the Release Wizard's Internet Options panel or the Create Default Web Page setting on the Internet tab in the Releases view. Then examine the code in the generated Web page (**Setup.htm**).

Once an instance of the Player has been created, you may use the methods and properties associated with each of the objects within the Player to customize your installation.

Passing Data to the Launched One-Click Install Installation

To pass data to a launched One-Click Install installation, use the is::CmdLine parameter and the CMDLINE script variable.

Running an Internet Installation Silently

To run an Internet installation silently, pass it the desired Setup.exe command-line options by calling the Ether object's SetProperty method, as in the following examples:

```
ether.SetProperty("is::CmdLine","-s -f1\"C:\\My Folder\\Mydir.iss\"");
```

```
ether.SetProperty("is::CmdLine","-s -f1\"C:\\My Folder\\Mydir.iss\" -f2\"C:\\My Folder\\Mydir.log\"");
```

Note the following:

- When running a silent Internet installation, you must specify the `-f1` option; there is no valid default location for the response file in Internet installations.
- When recording or running a silent Internet installation, you must specify a fully qualified absolute path if using the `-f1` or `-f2` option; these options do not accept URLs.
- The second argument to **SetProperty** must use the escape sequences (\\ and \") to specify backslashes and quotation marks, as in the preceding examples.

Setting the Run-Time Language



Project • *This information applies to InstallScript projects.*



Edition • *Multilingual support is available in InstallShield Premier Edition.*

When you specify a language by using the following method, the One-Click Install installation runs in this language regardless of the default language specified in InstallShield or the default language of the target system. If you do not use this method, the One-Click Install installation determines the run-time language in the same way as any other installation.



Task ***Add set the run-time language:***

1. Add support for the language to your project:
   a. On the **Project** menu, click **Settings**. The **Project Settings** dialog box opens.
   b. Click the **Languages** tab.
   c. Select the check box for that language that your installation should target.
2. Add that language to the release:
   a. On the **Build** menu, click **Release Wizard**.
   b. In the **Setup Languages** panel, select the appropriate language.

3. On your Web page, before you call the Play method, call the SetProperty method to specify the appropriate language identifier.

   For example, the language ID for English is 0009. To set English as the runtime language use the following code:

   ```
   ether.SetProperty("is::CmdLine","-l0009");
   ```

Running a One-Click Install Installation in Debug Mode



Project *• This information applies to InstallScript projects.*

To run your One-Click Install installation in debug mode, enter the following code before invoking ether.Play:

```
ether.SetProperty( "is::CmdLine", "/d")
```

Authenticating a User from InstallScript



Project *• This information applies to InstallScript projects.*

If your One-Click Install installation requires authentication after it is started, you can authenticate the user on the fly using InstallScript. When authentication is required, the Web server returns HTTP error code 401 to your installation, which generates an InternetError event. Respond to that event by prompting the user for a user name and password in the OnInternetError event handler.

The easiest way to do so is to use the **InternetErrorDlg** API in **WinInet.dll** and return the value IDRETRY from OnInternetError. This causes the installation to issue the HTTP request for the file again. If users enter incorrect information, they will be prompted to enter it again.

Setup.ini



Project *• This information applies to InstallScript projects.*

Setup.ini is an initialization file that is created during the build process of InstallScript-based projects to control certain elements of the installation. The build process fills in only certain key names and values in **Setup.ini**. After the build process creates **Setup.ini**, it is placed in the **Disk1** folder in **<project folder>\Media\<media name>\Disk Images**.

If you want to customize **Setup.ini** further, modify it with a text editor. You can automate this process by using the Postbuild Options panel of the Release Wizard or the Execute Batch File property in the Releases view to launch a batch file or executable file that performs the desired modifications. Do not simply overwrite **Setup.ini** with a modified copy from a previous media build; doing so could cause your installation to work improperly.

Setup.ini contains two predefined sections:

- [Startup]
- [Mif]

You can add additional sections to **Setup.ini** to pass information to your setup script. You can then call the **GetProfString** and **GetProfInt** functions to transfer the information from the **Setup.ini** file to your installation.

Here is an example **Setup.ini** file:

```
[Startup]
EnableLangDlg=Y
Product=My Application
ProductGUID=23EAFFCA-361D-11D3-8B0F-00105A9846E9
CompanyName=My Company Name
Skin=Setup.skin
SmallProgress=N
SplashTime=5
CheckMD5=N
CmdLine=/f1Test1.iss
LauncherName=MyInstall.exe

[Mif]
Type=SMS
Filename=Ishield
SerialNo=IS50-32XYZ-12345
Locale=DEU
```



***Note** • If you need to access the file later (after* **Disk1** *has been removed), you should copy* **Setup.ini** *to the support folder (SUPPORTDIR) at the beginning of the installation.*

The [Startup] Section

You can use the following keynames in the `[Startup]` section of **Setup.ini**:

- EnableLangDlg
- Product
- ProductGUID
- CompanyName
- Skin
- SmallProgress
- SplashTime
- CheckMD5
- CmdLine

- LauncherName

EnableLangDlg

The EnableLangDlg keyname tells the installation whether to display the Language dialog during installation initialization. The Language dialog lets your end user decide which available language the installation should run in. You can set this value in the **Setup Languages** panel of the **Release Wizard** or through the **Languages Dialog** property in the **Releases** view.

For more information about the Language dialog, see How an Installation Determines Which Language to Use for the User Interface.

Product

The Product keyname identifies an application or product name to be displayed at the beginning of the text string in the startup message dialog box.

ProductGUID

The ProductGUID keyname specifies the installation's GUID so that **Setup.exe** has access to this data during initialization. The media builder automatically enters the correct value in **Setup.ini**; do not change the value.

CompanyName

The CompanyName keyname specifies the name of the company name.

Skin

The Skin keyname specifies the name of the skin file that the installation uses to display end-user dialogs. If this keyname is not in **Setup.ini**, no skin is used. You can specify a skin in the User Interface panel of the Release Wizard or through the Specify Skin property in the Releases view.

SmallProgress

The SmallProgress keyname specifies whether the installation initialization dialog is the small box that was shown by installations created with InstallShield Professional version 6.31 and earlier, or if it is larger and is consistent with the rest of the end-user dialogs. Set the value to Y to display the small dialog or N to display the larger dialog. (If the installation displays a security, Save and/or Run Setup, Choose Setup Language, or Qualifying Product(s) Detected dialog, the installation initialization dialog is larger and is consistent with the rest of the end-user dialogs, regardless of the value of this key.) You can set this value in the User Interface panel of the Release Wizard or through the Small initialization dialog property in the Releases view.

If SmallProgress is set to N, the installation initialization dialog (and the Choose Setup Language dialog, if any) are not displayed until the startup graphic closes. To specify the length of time for which the startup graphic displays, set the SplashTime value.

SplashTime

The SplashTime keyname specifies the length of time (in seconds) for which the startup graphic is displayed.

CheckMD5

The CheckMD5 keyname tells the installation whether to compare each file's MD5 hash value to the value stored in the installation header file during file extraction. You can set this value with the Verify MD5 signature check box in the General Options - Advanced dialog box.



***Tip** • MD5 checking can detect corrupted files, which is useful during Internet installations; not doing MD5 checking can make file transfer proceed faster.*

CmdLine

The CmdLine keyname identifies command-line parameters that are to be passed to **Setup.exe** if none are passed from the command line.



Note** • If you use the `/v` parameter at the command prompt when launching* **Setup.exe, any parameters that are specified for the CmdLine keyname are ignored.*

For more information about available command-line parameters, see Setup.exe and Update.exe Command-Line Parameters.

LauncherName

If you rename **Setup.exe**, the LauncherName keyname specifies the file's new name. This keyname is required if another installation will launch this installation using the **DoInstall** function.

The [Mif] Section

If the `[Mif]` section is present, the installation will automatically create a installation .mif file in the **Temp** folder. You can also create an .mif file by using the `-m` command-line option for **Setup.exe** and optionally its `-m1` and `-m2` options.

These are the keynames under the `[Mif]` section of **Setup.ini**:

- Type
- Filename
- SerialNo
- Locale

Type

Set this key to SMS.

Filename

The Filename key is optional. It provides the alternate name for the .mif file to be created. If this key is not included, the installation tries to use the AppName key under the `[Startup]` section of **Setup.ini** as the .mif file name. If the AppName key is also not present, the installation creates a file with the default name **Status.mif**.

The file name should not include an extension, since .mif files must have the .mif extension. The file name should not include a path—it is placed in the **Temp** folder by default.

SerialNo

The SerialNo key is also optional. If provided, the information from this key is placed in the Serial Number section in the .mif file. If this key is not present, the installation instead places quotation marks ("").

Locale

The Locale key tells the installation to place the indicated locale in the .mif file. English (ENU) is the default; refer to Microsoft documentation for a complete listing of locale strings.

The following is an example of a **Setup.ini** file for an installation that will automatically create an .mif file.

```
[Startup]
AppName=InstallShield


[Mif]
Type=SMS
Filename=IShield
SerialNo=IS50-32XYZ-12345
Locale=DEU
```

Cloning Releases



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

InstallShield enables you to clone or copy a release in the Releases view. This allows you to create a release with all of its properties populated as they were in the cloned release. This way, if you want to create more than one release with only a few differences between them, you can clone a release and change only the settings that you need to change.



Task ***To copy or clone a release:***

1. On the View List under **Media**, click **Releases**.
2. Right-click the release that you want to copy and click **Clone**.

A release named *Copy of Release x* is created. The release copy has all of the same properties as the original release. Rename the release and modify the release settings that you want to change.

Creating UWP App Packages

Project *• UWP app creation is available in Basic MSI projects.*

Important *• The Windows 10 Anniversary Update is required for installing and testing a UWP app package (.appx) with desktop extensions (Desktop Bridge) included. In addition, InstallShield must be installed on a Windows 10 machine or a machine with the Windows 10 SDK installed in order to: digitally sign the UWP app package, utilize UWP app logo variants (described in UWP App Logo Considerations, or to build localized UWP app packages.*

The UWP app package (.appx) format is the simple and secure packaging format used to distribute and install apps on Windows 8.x and 10 and is the only format allowed for Universal Windows Platform (UWP) apps. Benefits of UWP app packages include:

- High availability, reliability, and durability, resulting in applications that operate continuously without failure for extended periods of time
- A smooth installation experience through static builds that require minimal configuration and no customizable UI
- The option to sell or provide the application through the Windows Store
- The ability to leverage UWP functionality such as live tiles as well as the ability to utilize UWP APIs
- The only package format with native support on Windows Nano Server

InstallShield supports creating the UWP app package format and its desktop and server extensions and also provides testing to help you identify items unsuitable for the UWP app package format. When you select a release in the Releases view, a per-release tab titled **Windows App** includes settings to create a UWP app package. Here, various core settings can be specified that impact the UWP app package build process.

UWP app packages can be localized using the settings in the **Display Properties** area on the **Windows App** tab. The supported languages used for the UWP app package build match those in the **UI Languages** settings on the Build tab of the release, as long as **MakePRI.exe** (within the Windows 10 SDK) is found on the build machine. If **MakePRI.exe** cannot be found on the build machine, InstallShield will only generate a UWP app package in the default language.

The following instructions provide details on how to create a UWP app package.

Task ***To convert a Basic MSI project to UWP app package (.appx) format:***

1. Open the Basic MSI Project that you want to build a UWP app package for as part of building the .msi release.
2. In the **Releases** explorer, under the product configuration, click the name of the release for which you want to build a UWP app package.
3. Click the **Windows App** tab and in the **Build UWP App Package** setting, specify **Yes**.
4. Do the following:
   - Use the **General** area on the **Windows App** tab to specify core settings that impact the build process such as choosing whether to include desktop extensions or server extensions.

- Use the **Package Identity Overrides** area on the **Windows App** tab to uniquely identify the package to Windows.
- Use the **Display Properties** area on the **Windows App** tab to identify the package to an end user.

For descriptions of these settings, refer to the Windows App Tab for a Release topic.

5. To configure tiles created in the UWP app package, open the **Shortcuts** view and configure settings in the **UWP App Package Tile Overrides** area. For descriptions of the **UWP App Package Tile Overrides** settings, refer to the Shortcut Settings topic.
6. Build the release.
7. To scan the .msi package for signs of items that are unsuitable for the UWP app package format, run the InstallShield UWP App Suitability Suite. On the **Build** menu, point to **Validation**, and then click **InstallShield UWP App Suitability Suite.** For more information, refer to the InstallShield UWP App Suitability Suite topic.

   For information about how to deploy the UWP app package, refer to Deploying UWP App Packages.

Testing and Running Installations

Before distributing your installation, you should test run it to ensure that you discover any issues rather than your customers. InstallShield enables you to test the end-user dialogs (without copying any files to the target system), run the entire installation including transferring files.

Test Running Installations

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

You can test run an installation to review the end-user dialogs. The test run displays the full user interface, but it does not copy any files over to the target system or update the registry.

Task ***To test run a project, do one of the following:***

- In the **Releases** view, right-click a release and click **Test Setup**.
- Click the **Test** button on the toolbar.

Running Installations from the InstallShield Interface

After you run the Release Wizard and build a release of your installation, you can run it from within InstallShield.



Task ***To run any of your built installation packages, do one of the following:***

- Click the **Run** button on the toolbar.
- In the **Releases** view, right-click a release and click **Run Setup**.
- On the **Build** menu, click **Run ReleaseName**, where *ReLeaseName* is the name of the release that currently has focus in the **Releases** view.

InstallShield runs the release that currently has focus in the Releases view.

If you selected the Uninstall before installing check box in the **Options** dialog box, the previously run release (if any) is uninstalled before the current release is run. This option is available on the Preferences tab in the Options dialog box.

You cannot specify any command-line parameters when launching your installation from the InstallShield interface. You must launch your installation from the command line to pass parameters to the Windows Installer. For more information, see MsiExec.exe Command-Line Parameters.

Running Installations in Silent Mode

Silent installations are installations that run without an end-user interface. If you want your installation to run silently, InstallShield allows you to create silent installations for Basic MSI, InstallScript MSI, and InstallScript project types.

Basic MSI Silent Installations

To run a Basic MSI installation silently, type the following at the command line:

```
msiexec /i Product.msi /qn
```

If your release settings include **Setup.exe**, you can run the following command:

```
Setup.exe /s /v"/qn"
```

Basic MSI installations do not create or read response files. To set installation properties for a Basic MSI project, run a command line such as:

```
msiexec /i Product.msi /qn INSTALLDIR=D:\ProductFolder USERNAME="Valued Customer"
```

InstallScript MSI and InstallScript Silent Installations

For InstallScript MSI and InstallScript projects, you need to record a response file that records the end-user interaction. This response file is passed to **Setup.exe** so that the installation can be run. The traditional silent installation works almost exactly the same as regular installations. It follows the same script logic as the regular installation.

If you need to install an InstallScript MSI installation without using **Setup.exe**, you can use the MSI silent mode.

InstallScript MSI and InstallScript Silent Installations



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

InstallShield Silent allows automated electronic software distribution, also known as silent installation. With InstallShield Silent, there is no need for end users to monitor the installation and provide input through dialogs. An InstallShield Silent installation runs on its own, without any end-user intervention.

If there are multiple instances of your application on a machine, and if a silent update installation for your application runs on that machine, it applies the update to the first instance that it finds, and it does not display the Qualifying Product(s) Detected dialog.

To launch InstallShield Silent, use the **Setup.exe** `-s` command line parameter.



***Windows Logo** • To comply with Windows logo requirements, a silent installation must create a response file in which the default installation options are selected.*

You can run your installation with the **Setup.exe** `-r` parameter to select installation options and automatically record the InstallShield Silent response file, or you can create your own. To view a real-world example of a response file, refer to the **Setup.iss** file located on InstallShield installation's **Disk1**. For a description of the response file format, see Creating the Response File.

If you need to install an InstallScript MSI installation without using **Setup.exe**, you can use the MSI silent mode.



Task ***To create a silent installation:***

1. Create the installation.
2. Create the response file.
3. Run the silent installation.
4. Check for errors.

Creating a Silent Installation



***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

If you want to create an installation that will be run in silent mode, create your installation in a typical manner, and test the script in the normal (non-silent) manner.

You can easily modify your installation script logic to include flow control based on whether your installation is being run in silent mode. InstallShield provides a system variable called *MODE*. The *MODE* variable contains one of the following constants to identify the installation's current mode:

- **NORMALMODE**—Indicates that the installation is running in normal mode.
- **RECORDMODE**—Indicates that the **Setup.exe** file is automatically generating a silent installation file (.iss file), which is a record of the installation input, in the Windows folder.

- **SILENTMODE**—Indicates that the installation is running in silent mode.

For more information, see MODE.



***Tip** • All InstallShield built-in and Sd dialogs automatically handle the values stored in the InstallShield Silent response file (.iss file). If you are creating custom dialogs, you will need to call **SilentReadData** to handle the dialog's return values in silent mode.*

After you have created or modified the installation, the next step in creating a silent installation is to create the response file.

Creating the Response File



***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

A normal (non-silent) installation receives the necessary input from end users in the form of responses to dialogs. However, a silent installation does not prompt the end user for input. A silent installation must get its end-user input from a different source. That source is the InstallShield Silent response file (.iss file).

A response file contains information similar to that which an end user would enter as responses to dialogs when running a normal installation. InstallShield Silent reads the necessary input from the response file at run time.

The format of response files resembles that of an .ini file, but response files have .iss extensions. A response file is a plain text file consisting of sections containing data entries.

There are two ways in which you can create an InstallShield Silent response file: you can run the installation and have InstallShield record and create the response file for you, or you can write the response file directly.

Recording a Response File

You have the option of letting InstallShield create the response file for you. Simply run your installation with the **Setup.exe** `-r` command-line parameter. InstallShield will record all your installation choices in **Setup.iss** and place the file in the Windows folder.

All InstallShield built-in and Sd dialog functions are designed to write values into the **Setup.iss** file when InstallShield runs in record mode (Setup -r). If you are creating custom dialogs, you will need to call **SdMakeName** and **SilentWriteData** to add sections and dialog data to the response file when the installation runs in record mode. Refer to the Sd dialogs' source code in the **<InstallShield location>\Include** folder for examples of using these functions to write to **Setup.iss**. Read the following section for more information about what data to add to **Setup.iss** when calling **SdMakeName** and **SilentWriteData**.

Manually Creating a Response File

You can also create the response file completely by hand. As mentioned, the **Setup.iss** file is similar to an .ini file. The sections of an InstallShield response file must be in the following order:

1. Silent Header Section
2. Application Header Section
3. Dialog Sequence Section
4. Dialog Data Sections (one per dialog)

Section names are contained in square brackets, as in **[InstallShield Silent]**.

Data entries follow their section names and consist of <name=value> pairs, as in the following example:

```
Dlg0={23EAFFCA-361D-11D3-8B0F-00105A9846E9}-Welcome-0
```



Task ***To create the response file:***

1. Create a text file named **Setup.iss** using any text editor.
2. Enter the silent header into **Setup.iss**.
3. Enter the application header into **Setup.iss**.
4. Enter the dialog sequence into **Setup.iss**.
5. Enter the dialog data into **Setup.iss**.
6. Save and close **Setup.iss**.

A sample response file is included to help familiarize you with the format.

Response File Silent Header



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

All response files begin with a response file silent header. The response file silent header enables InstallShield to identify the file as a legitimate InstallShield response file. It also helps to verify that the response file corresponds to an installation created with the proper version of InstallShield.

The format of the silent header is shown below. Enter the following lines at the beginning of your **Setup.iss** file:

```
[InstallShield Silent]
Version=v7.00
File=Response File
```

The Version=v7.00 line indicates the version of the InstallShield Silent response file, not the version of InstallShield. Use **v7.00** in all response files.

When you are creating response files, be sure to use the same version of InstallShield to create the response file that will be used to run the silent installation.

Response File Application Header



***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The response file application header is the second block of information in the response file, immediately following the response file silent header. The response file application header enables you to identify response files visually. It is not used by the installation script or by **Setup.exe**. You can use the application header to identify exactly which installation the response file is for, since it is often difficult to determine this by looking at the dialog data.

The format of the application header is shown below. The values assigned to Name, Version, and Company are derived from the values written to the registry in the call to the **CreateInstallationInfo** function in your installation script. (In an event-based script, **CreateInstallationInfo** is called in the default OnMoveData event handler code.) Enter the following lines into your **Setup.iss** file below the silent header:

```
[Application]
Name=<ProductKey from CreateInstallationInfo>
Version=<VersionKey from CreateInstallationInfo>
Company=<CompanyKey from CreateInstallationInfo>
```

Response File Dialog Sequence



***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The third block of information in the response file, after the silent header and the application header, is the response file dialog sequence. The dialog sequence section lists all dialogs that you would need to use in a normal installation (including custom dialogs), in the order in which they would appear. The dialogs are listed under the section heading [<PRODUCT_GUID>-DlgOrder].

The dialog numbering sequence begins at 0. There is no limit to the number of dialogs that you can list.

The order and the number of dialogs is significant. When InstallShield Silent runs, if either the number or the order of the dialogs does not match the order or the number of the dialogs in the non-silent installation, the silent installation fails and the log file records the failure. Make one entry for each occurrence of a dialog. A given dialog may be listed more than once if it appears more than once in the installation.

The format for a dialog sequence entry is as follows:

```
Dlg<#>=<PRODUCT_GUID>-<DialogIdentifier>
```

The Dlg<#> part consists of the letters Dlg and a sequence number. The first dialog in the installation is Dlg0. Each dialog after that increments the value of <#> by one.

<DialogIdentifier> is in the form *functionname*-<#>, where *functionname* is the name of the function that created the dialog, and <#> represents the sequential order of that particular dialog in the installation.

For custom dialogs, you can use any unique alphanumeric name for the *functionname* portion of <DialogIdentifier>. For example, you could refer to a custom dialog as MyDialog. If the tenth dialog in the installation were the second occurrence of the custom dialog MyDialog, there would be an entry in the dialog sequence section such as the following:

```
Dlg9=<PRODUCT_GUID>-MyDialog-1
```

The <DialogIdentifier> expression is used to identify the dialog data section for the dialog.

Always end the dialog sequence section with a statement that has the following form:

```
Count=<number of dialogs>
```

The statement specifies the exact number of dialogs listed in the dialog sequence section. Count every entry. Since dialog numbering begins with 0, the value of Count will always be 1 greater than the highest <#> value for a dialog sequence.

The example below lists two dialogs, the Welcome dialog and the **AskOptions** dialog. Enter your dialog sequence list into **Setup.iss** as shown in the example below.

```
[{23EAFFCA-361D-11D3-8B0F-00105A9846E9}-DlgOrder]
Dlg0={23EAFFCA-361D-11D3-8B0F-00105A9846E9}-Welcome-0
Dlg1={23EAFFCA-361D-11D3-8B0F-00105A9846E9}-AskOptions-0
Count=2
```

Response File Dialog Data



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The last block of information in a response file is the response file dialog data. The response file dialog data is a collection of sections containing the values returned by each dialog identified in the dialog sequence section. Each dialog has its own section. The values listed are the same values that the dialog returns in a normal end-user input-driven installation. You can also create dialog data sections for custom dialogs.

The format for a dialog data section is shown below:

```
[<PRODUCT_GUID>-<DialogIdentifier>]
Result=value
Keyname1=value
Keyname2=value
```

The [<PRODUCT_GUID>-<DialogIdentifier>] section header identifies the specific dialog and is followed by the dialog data entry list. <DialogIdentifier> is the same expression used to list the dialog in the dialog sequence section.

Data entry items are in the format keyname=value. The keyname is a name for a value returned by a dialog and recorded in the response file. All dialogs, including custom dialogs, return a value for the keyname Result, reflecting the button that was clicked to end or exit the dialog. Many dialogs will also set or return a value in a variable.

For example, in a non-silent installation, the **AskDestPath** dialog returns the destination location in the `svDir` parameter. The line in the script might look like the following:

```
nResult = AskDestPath( szTitle, szMsg, svDir, 0 );
```

The corresponding dialog data entry in the **Setup.iss** file for a silent installation might look like the following:

```
[{23EAFFCA-361D-11D3-8B0F-00105A9846E9}-AskDestPath-0]
Result=1
szPath=C:\Program Files\InstallShield\2016
```

In the above example, **Result=1** is equivalent to clicking the Next button in the dialog, and **szPath=C:\Program Files\InstallShield\2016** is the value returned in the `svDir` parameter of **AskDestPath**.

The name of the variable used in the script is meaningless relative to the **Setup.iss** file. However, in the **Setup.iss** dialog data sections, each built-in and Sd dialog has its own set of keynames that map to its parameters. The keynames are important and must be exactly as defined for each dialog. Refer to Dialog Data Keynames List, below.

When you use custom dialogs, you must create a dialog data entry for the Result keyname for each dialog, plus entries for any other values set or returned by the custom dialogs. Use the Dialog Data Keynames List, below, as an indicator of how to create keyname=value expressions for your custom dialogs. Call **GetProfString** or **GetProfInt** to read the dialog data information for the custom dialog. **GetProfString** and **GetProfInt** allow you to specify the .iss file, the section, and the keyname, and they return the value assigned to that keyname.

Standard Values for the Result Keyname

All dialogs, including custom dialogs, return a keyname value called Result, indicating which button was clicked to end the dialog. Unless otherwise indicated, the Result standard values are:

- 12 for the Back button
- 1 for the Next or OK button

Recording Component and Subcomponent Selections

Some dialog functions allow the end user to select components and subcomponents. There are three kinds of dialog data keyname entries used to record component and subcomponent selections in response files: type, count, and <#> (described below).

Every set of component selections and every set of subcomponent selections has one type keyname entry, one count keyname entry, and as many <#> keyname entries as are required to document each individual component or subcomponent selection.

When you are creating keynames to record component selections, precede the type, count, and <#> keyname entries with the word Component, thus:

```
Component-typeComponent-countComponent-0
```

When you are creating keynames to record subcomponent selections, precede the type, count, and <#> keyname entries with the name of the component to which the subcomponents belong, thus:

```
Program Files-typeProgram Files-countProgram Files-0Program Files-1
```

To create complete value entries, use the equal sign to attach the values to the keynames. (The types of values assigned to each kind of keyname are described below.) The following example shows complete value entries for two components, **Program Files** and **Binary Files**, and two subcomponents of **Program Files**, **Executables** and **Support Elements**:

```
Component-type=string
Component-count=2
Component-0=Program Files
Component-1=Binary Files
Program Files-type=string
Program Files-count=2
Program Files-0=Executables
Program Files-1=Support Elements
```

The type keyname indicates the data type of the components or subcomponents list. Since InstallShield dialogs currently use only string lists for components and subcomponents lists, type is always equal to **string**, as in Component-type=string. Future versions may use number lists, in which case type could equal **number**.

Count Keyname Entry

Count is equal to the number of selections for a given component or subcomponent entry. For example, if two components were selected for installation in the **ComponentDialog** dialog, the count dialog data entry would be **Component-count=2**. If two subcomponents of the **Program Files** component were selected, there would be a **Program Files-count=2** entry.

<#> Keyname Entry

The number portion of the <#> keyname entry is simply a sequential (one-up) number, beginning with 0, that numbers each recorded component or subcomponent selection. Since numbering begins with 0, the greatest number value will always be one less than the value of count.

The value assigned to a <#> keyname entry is the selected component's or subcomponent's visible name (the string passed as the second parameter to **ComponentAddItem** when the components or subcomponents list was built).

For example, assume the **ComponentDialog** dialog offers end users a component selection of **Program Files**, **Help Files**, **Sample Files**, and **Utilities**. If the end user selects **Program Files** and **Help Files**, then the dialog data section for that instance of the **ComponentDialog** dialog will have two list item entries and will look something like this:

```
[{23EAFFCA-361D-11D3-8B0F-00105A9846E9}-ComponentDialog-0]
szDir=C:\MYAPP\FILES
Component-type=string
Component-count=2
Component-0=Program Files
Component-1=Help Files
Result=
```

The following example shows how subcomponent list selections are recorded. The example is for an instance of the **SdComponentMult** dialog. The example shows that two components—**Program Files** and **Help Files**—were selected. It also shows that four subcomponents were chosen: **Main Files**, **Aux. Files**, **Main Help**, and **Tutorial Files**. **Main Files** and **Aux. Files** are subcomponents of **Program Files**, and **Main Help** and **Tutorial Files** subcomponents of **Help Files**.

```
[{23EAFFCA-361D-11D3-8B0F-00105A9846E9}-SdComponentMult-0]
Component-type=string
Component-count=2
Component-0=Program Files
Component-1=Help Files
Program Files-type=string
Program Files-count=2
Program Files-0=Main Files
Program Files-1=Aux. Files
Help Files-type=string
```

```
Help Files-count=2
Help Files-0=Main Help
Help Files-1=Tutorial Files
Result=1
```

Dialog Data Keynames List

The dialog data keynames for the InstallShield dialogs are listed in the table below. The first column contains the dialog names. The second column lists the keynames applicable to each dialog. The third column contains descriptions of the values associated with the keynames.

Table 4-16 • Dialog Data Keynames

Dialog	Keyname	Description
AskDestPath-<#>	Result	Standard values
AskDestPath-<#>	szPath	The path that is set in the edit field
AskOptions-<#>	Result	Standard values AskOptions is special because the number of return values can change based on the script call. You can use a sequence of keynames of the form Sel-<#>, where <#> is the number of the selection variable. Numbering begins with 0. The number of Sel-<#> entries should match the number of variables (check boxes/radio buttons) in the particular call to **AskOptions**. Refer to the following examples: • **Sel-0**—Maps to first selection variable in **AskOptions**. • **Sel-1**—Maps to second selection variable in **AskOptions**. • **Sel-2**—Maps to third selection variable in **AskOptions**.
AskPath-<#>	Result	Standard values
AskPath-<#>	szPath	The path entered in the Destination Directory edit field
AskText-<#>	Result	Standard values
AskText-<#>	szText	The text from the edit field
AskYesNo-<#>	Result	1 = End user clicked the Yes button 0 = End user clicked the No button
ComponentDialog-<#>	Result	Standard values
ComponentDialog-<#>	szDir	The path entered in the Destination Directory field
ComponentDialog-<#>	Component-type	String (the only value currently allowed)

Table 4-16 • Dialog Data Keynames (cont.)

Dialog	Keyname	Description
ComponentDialog-<#>	Component-count	The total number of component selections
ComponentDialog-<#>	Component-<#>	The selected item's number in the list (numbering begins with 0)
MessageBox-<#>	Result	1 = End user clicked the OK button
RebootDialog-<#>	Result	0 (Result is always 0.)
RebootDialog-<#>	Choice	Indicates the final reboot selection made before restarting the machine. Maps to the radio buttons and has these possible values: • 602 = "Yes, I want to restart my computer now." • 0 = "No, I will restart my computer later."
SdAskDestPath-<#>	Result	Standard values
SdAskDestPath-<#>	szDir	The path entered in the Destination Directory field
SdAskOptions-<#>	Result	Standard values
SdAskOptions-<#>	Component-type	String (the only value currently allowed)
SdAskOptions-<#>	Component-count	The total number of component selections
SdAskOptions-<#>	Component-<#>	The selected item's number in the list (numbering begins with 0)
SdAskOptionsList-<#>	Result	Standard values
SdAskOptionsList-<#>	Component-type	String (the only value currently allowed)
SdAskOptionsList-<#>	Component-count	The total number of component selections
SdAskOptionsList-<#>	Component-<#>	The selected item's number in the list (numbering begins with 0)
SdBitmap-<#>	Result	Standard values
SdComponentDialog-<#>	Result	Standard values

Table 4-16 • Dialog Data Keynames (cont.)

Dialog	Keyname	Description
SdComponentDialog-<#>	szDir	The path entered in the Destination Directory field
SdComponentDialog-<#>	Component-type	String (the only value currently allowed)
SdComponentDialog-<#>	Component-count	The total number of component selections
SdComponentDialog-<#>	Component-<#>	The selected item's number in the list (numbering begins with 0)
SdComponentDialog2-<#>	Result	Standard values
SdComponentDialog2-<#>	Component-type, <subcomponent>-type	String (the only value currently allowed)
SdComponentDialog2-<#>	Component-count, <subcomponent>-count	The total number of component or subcomponent selections
SdComponentDialog2-<#>	Component-<#>, <subcomponent>-<#>	The selected item's number in the list (numbering begins with 0)
SdComponentDialogAdv-<#>	Result	Standard values
SdComponentDialogAdv-<#>	szDir	The path entered in the Destination Directory field
SdComponentDialogAdv-<#>	Component-type	String (the only value currently allowed)
SdComponentDialogAdv-<#>	Component-count	The total number of component selections
SdComponentDialogAdv-<#>	Component-<#>	The selected item's number in the list (numbering begins with 0)
SdComponentMult-<#>	Result	Standard values

Table 4-16 • Dialog Data Keynames (cont.)

Dialog	Keyname	Description
SdComponentMult-<#>	Component-type, <subcomponent>-type	String (the only value currently allowed)
SdComponentMult-<#>	Component-count, <subcomponent>-count	The total number of component or subcomponent selections
SdComponentMult-<#>	Component-number, <subcomponent>-<#>	The selected item's number in the list (numbering begins with 0)
SdComponentTree-<#>	Result	Standard values
SdConfirmNewDir-<#>	Result	1 = End user clicked the Yes button 0 = End user clicked the No button
SdConfirmRegistration-<#>	Result	1 = End user clicked the Yes button 0 = End user clicked the No button
SdDisplayTopics-<#>	Result	Standard values
SdFinish-<#>	Result	1 = Finish
SdFinish-<#>	bOpt1	1 = "Yes, I want to view the README file." is selected 0 = "Yes, I want to view the README file." is not selected
SdFinish-<#>	bOpt2	1 = "Yes, I want to launch [app name]" is selected 0 = "Yes, I want to launch [app name]" is not selected
SdFinishEx		Requires special handling.
SdFinishReboot-<#>	Result	1 = Finish
SdFinishReboot-<#>	BootOption	0 = Do not restart Windows or the machine 2 = Restart Windows 3 = Reboot the machine
SdLicense-<#>	Result	12 = End user clicked the Back button 1 = End user clicked the Yes button

Table 4-16 • Dialog Data Keynames (cont.)

Dialog	Keyname	Description
SdOptionsButtons-<#>	Result	12 = End user clicked the Back button Or, when the Next button is clicked: 101 = Option button one (top) is selected 102 = Option button two is selected 103 = Option button three is selected 104 = Option button four is selected
SdRegisterUser-<#>	Result	Standard values
SdRegisterUser-<#>	szName	The text entered in the Name field
SdRegisterUser-<#>	szCompany	The text entered in the Company field
SdRegisterUserEx-<#>	Result	Standard values
SdRegisterUserEx-<#>	szName	The text entered in the Name field
SdRegisterUserEx-<#>	szCompany	The text entered in the Company field
SdRegisterUserEx-<#>	szSerial	The text entered in the Serial (number) field
SdSelectFolder-<#>	Result	Standard values
SdSelectFolder-<#>	szFolder	The folder name entered in the Program Folder field
SdSetupType-<#>	Result	12 = End user clicked the Back button Or, when the Next button is clicked: 301 = Typical radio button is currently selected 302 = Compact radio button is currently selected 303 = Custom radio button is currently selected
SdSetupType-<#>	szDir	The path entered in the Destination Directory edit field
SdShowDlgEdit1-<#>	Result	Standard values
SdShowDlgEdit1-<#>	szEdit1	The text entered in the svEdit1 field
SdShowDlgEdit2-<#>	Result	Standard values
SdShowDlgEdit2-<#>	szEdit1	The text entered in the svEdit1 field
SdShowDlgEdit2-<#>	szEdit2	The text entered in the svEdit2 field

Table 4-16 • Dialog Data Keynames (cont.)

Dialog	Keyname	Description
SdShowDlgEdit3-<#>	Result	Standard values
SdShowDlgEdit3-<#>	szEdit1	The text entered in the svEdit1 field
SdShowDlgEdit3-<#>	szEdit2	The text entered in the svEdit2 field
SdShowDlgEdit3-<#>	szEdit3	The text entered in the svEdit3 field
SdShowFileMods-<#>	Result	Standard values
SdShowFileMods-<#>	nSelection	The "Choose what you want Setup to do" radio button selection: 101 = Let Setup modify the AUTOEXEC.BAT file. 102 = Save the required changes to AUTOEXEC.NEW file. 103 = Do not make any changes.
SdShowInfoList-<#>	Result	Standard values
SdStartCopy-<#>	Result	Standard values
SdWelcome-<#>	Result	Standard values
SdWelcomeMaint-<#>	Result	301 = Indicates that the Modify button was selected when the Next button was clicked. 302 = Indicates that the Repair button was selected when the Next button was clicked. 303 = Indicates that the Remove button was selected when the Next button was clicked.
SelectDir-<#>	Result	Standard values
SelectDir-<#>	szDir	The path selected by this dialog
SelectDirEx-<#>	Result	Standard values
SelectDirEx-<#>	szDir	The path selected by this dialog
SelectFolder-<#>	Result	Standard values
SelectFolder-<#>	szResultFolder	The folder selected by this dialog

Table 4-16 • Dialog Data Keynames (cont.)

Dialog	Keyname	Description
SetupType-<#>	Result	12 = End user clicked the Back button Or, when the Next button is clicked: 301 = Typical radio button is currently selected 302 = Compact radio button is currently selected 303 = Custom radio button is currently selected
SprintfBox-<#>	Result	1 = End user clicked the OK button
Welcome-<#>	Result	Standard values

Special Handling for SdFinishEx

SdFinishEx calls either **SdFinish** or **SdFinishReboot**, depending on the value of the system variable *BATCH_INSTALL*. For this reason, *SdFinishEx* cannot be handled by a response file. To create a silent installation for a script that calls *SdFinishEx*, use script code such as the following:

```
if (MODE = NORMALMODE) then
    SdFinishEx (szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);
else
    if !BATCH_INSTALL then
        /* Set bvOpt1 and bvOpt2 to the values you want them
        to have during a silent installation that does not require
        rebooting; for example: */
        bvOpt1 = FALSE;
        bvOpt2 = TRUE;
    else
        /* When the silent installation requires rebooting,
        if you want to reboot immediately call
        the System function as follows: */
        System (SYS_BOOTMACHINE);
    endif;
endif;
```

Sample Response File



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The following response file is the .iss file for a silent InstallShield installation.

```
[InstallShield Silent]
Version=v7.00
File=Response File

[Application]
Name=InstallShield
```

```
Version=10.50.000
Company=My Software Company

[{77AB941D-5876-11D4-A4A2-006067620F66}-DlgOrder]
Dlg0={77AB941D-5876-11D4-A4A2-006067620F66}-SdBitmap-0
Count=8
Dlg1={77AB941D-5876-11D4-A4A2-006067620F66}-Welcome-0
Dlg2={77AB941D-5876-11D4-A4A2-006067620F66}-SdRegisterUser-0
Dlg3={77AB941D-5876-11D4-A4A2-006067620F66}-AskDestPath-0
Dlg4={77AB941D-5876-11D4-A4A2-006067620F66}-SetupType-0
Dlg5={77AB941D-5876-11D4-A4A2-006067620F66}-SelectFolder-0
Dlg6={77AB941D-5876-11D4-A4A2-006067620F66}-SdStartCopy-0
Dlg7={77AB941D-5876-11D4-A4A2-006067620F66}-SdFinish-0

[{77AB941D-5876-11D4-A4A2-006067620F66}-SdBitmap-0]
Result=1

[{77AB941D-5876-11D4-A4A2-006067620F66}-Welcome-0]
Result=1

[{77AB941D-5876-11D4-A4A2-006067620F66}-SdRegisterUser-0]
szName=John Doe
szCompany=My Software Company
Result=1

[{77AB941D-5876-11D4-A4A2-006067620F66}-AskDestPath-0]
szPath=C:\Program Files\Product Name Version
Result=1

[{77AB941D-5876-11D4-A4A2-006067620F66}-SetupType-0]
Result=301
szDir=C:\Program Files\Product Name Version

[{77AB941D-5876-11D4-A4A2-006067620F66}-SelectFolder-0]
szResultFolder=Product Name Version
Result=1

[{77AB941D-5876-11D4-A4A2-006067620F66}-SdStartCopy-0]
Result=1

[{77AB941D-5876-11D4-A4A2-006067620F66}-SdFinish-0]
Result=1
bOpt1=1
bOpt2=0
```

Running the Silent Installation



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

After you have created the installation and the response file, do the following:

1. Add the response file (which is located in your Windows folder by default) to the Disk1 folder under Advanced Files in the Support Files/Billboards view.
2. Build your release. If you are creating a self-extracting executable file for your installation, enter `-s` for the Setup Command Line property in the General Options panel of the Release Wizard or for the Setup Command Line property in the Releases view.

Now you are ready to run the installation in silent mode using InstallShield Silent. When running an installation in silent mode, be aware that no messages are displayed. Instead, a log file named **Setup.log** captures installation information, including whether the installation was successful. You can review the log file and determine the result of the installation. (Note that for certain installation initialization errors, the log file may instead be named **Setupexe.log** and be created in SUPPORTDIR if the installation is run from the Internet or in SRCDIR otherwise.)

To launch InstallShield Silent, run **Setup.exe** with the `-s` option. If you created a self-extracting executable file, simply launch it; you included the `-s` option in step 2 above.

InstallShield also provides the `-f1` and `-f2` switches so that you can specify the name and location of the response file and the location of the log file. For more information, see Setup.exe and Update.exe Command-Line Parameters.

To verify if a silent installation succeeded, look at the ResultCode value in the [ResponseResult] section of **Setup.log**. InstallShield writes an appropriate return value after the ResultCode keyname.

Checking for Errors Using the Setup.log File



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Setup.log is the default name for the silent installation log file—generated when the end user runs **Setup.exe** with the `/s` argument—and it is by default created in the directory containing the response file Setup.iss. You can specify a different name and location for **Setup.log** using the `/f1` and `/f2` switches to **Setup.exe**.

The **Setup.log** file contains three sections. The first section, [InstallShield Silent], identifies the version of InstallShield Silent used in the silent installation. It also identifies the file as a log file.

The second section, [Application], identifies the installed application's name and version, and the company name.

The third section, [ResponseResult], contains the result code indicating whether the silent installation succeeded. An integer value is assigned to the ResultCode keyname in the [ResponseResult] section. The installation places one of the following return values in the ResultCode key.



Project • *Note that in an InstallScript MSI installation, there is no initialization process for reading or writing response files. As a result, the only errors that may occur for InstallScript MSI installation are -3 for silent installations and -6 for recording installations. Thus, the following table shows applicable project types where appropriate.*

Table 4-17 • Return Values for the Setup.log File

Result Code	Project Type	Description
0	InstallScript, InstallScript MSI	Success.
-1	InstallScript	General error.
-2	InstallScript	Invalid mode.
-3	InstallScript, InstallScript MSI	Required data not found in the Setup.iss file.
-4	InstallScript	Not enough memory available.
-5	InstallScript	File does not exist.
-6	InstallScript, InstallScript MSI	Cannot write to the response file.
-7	InstallScript	Unable to write to the log file.
-8	InstallScript	Invalid path to the InstallShield Silent response (.iss) file.
-9	InstallScript	Not a valid list type (string or number).
-10	InstallScript	Data type is invalid.
-11	InstallScript	Unknown error during setup.
-12	InstallScript	Dialog boxes are out of order.
-51	InstallScript	Cannot create the specified folder.
-52	InstallScript	Cannot access the specified file or folder.
-53	InstallScript	Invalid option selected.

The **Setup.log** file for a successful silent installation appears as follows:

```
[Application]
Name=Sample App 3000
Version=1.00.0000
Company=Sample Software Corporation
Lang=0409
```

```
[ResponseResult]
ResultCode=0
```

MSI Silent Mode Installations for InstallScript MSI Projects



Project • *This information applies to InstallScript MSI projects.*

If you need to silently install an InstallScript MSI project without using **Setup.exe**, you can use the MSI silent mode.

Launching MSI Silent Installations for InstallScript MSI Projects

The MSI silent installation is launched in the following cases:

- The installation is launched at the command line by typing the following:

  ```
  msiexec product.msi /qn
  ```

- The installation is activated from an advertised shortcut.
- The installation is activated from install-on-demand.
- When the product is removed when an update package is running.

Unlike the traditional silent mode, the MSI silent mode does not follow the regular logic provided through script. It simply runs through the InstallExecuteSequence table. (To see this table, navigate to the Direct Editor.) As a result, some events are not called, including the following:

- All UI installation events—OnFirstUIBefore and OnFirstUIAfter
- All Feature events

The OnMsiSilentInstall Event

What the Event Does

In an InstallScript MSI–based installation, InstallShield runs the OnMsiSilentInstall event handler if the product is not already installed on the target system. This happens if the installation is activated from an advertised shortcut or if the installation is launched through the following command line:

```
msiexec product.msi /qn
```

You need to override the OnMsiSilentInstall event if you want to support the MSI silent installation mode. This allows you to perform tasks that are normally performed in the OnFirstUIBefore, OnFirstUIAfter, and feature event handlers.

Overriding the Event

By default, **OnMsiSilentInstall** displays a message and then aborts the installation. You can override this event handler by writing your own implementation of the function. The prototype of this function is as follows:

```
external prototype OnMsiSilentInstall(HWND hInstall);
```

where `hInstall` is the handle to the installation.

The simplest thing that you can do is to implement an empty body of this event so that the installation will not abort. For example:

```
function OnMsiSilentInstall(hInstall)
begin
  //Do nothing and allow installation to continue.
end;
```

Again, **OnMsiSilentInstall** will be called on MSI silent installation and on activation of an advertised shortcut. It will not be called on Install-On-Demand, auto-repair, and uninstall mode.

Detecting the Mode in which a Silent Installation is Running

To detect whether a traditional silent installation is running from InstallShield script, use the following:

```
if (MODE = SILENTMODE)
```

To detect whether an MSI silent mode installation (including `/q`, advertise, auto-repair, uninstall, or install-on-demand) is running from InstallShield script, check the Windows Installer property **ISSETUP_UISEQUENCE_PROCESSED** using the **MsiGetProperty** Windows Installer API function. If this property is not set, then it is a silent install. (It indicates that the InstallUISequence is not executed.)

Silent Uninstallations from the Command Line (InstallScript MSI Only)

Project • *This information applies to InstallScript MSI projects.*

To initiate a silent uninstallation for a product that was installed through an InstallScript MSI installation, use a response file.

Task ***To run an uninstallation using a response file:***

1. Prepare a response file for the uninstallation (.iss) by running **Setup.exe** with the `/r` argument:

   ```
   Setup.exe /r
   ```

2. Type the following at the command line (items in Italics represent data that is specific to your product's uninstallation):

   ```
   Setup.exe /s /f1"FullPath\YourResponseFile.iss"
   ```

Validating Projects

Validating a project involves applying a set of internal consistency evaluator (ICE) rules to your installation or merge module package. The ICEs are designed to help you determine whether the package contains a valid database that performs its actions correctly. If a package fails one or more ICEs, InstallShield reports the specific ICE rules that were violated and offers additional information to help you troubleshoot the problem.

Microsoft created many of the ICEs that are available in InstallShield; Flexera Software created the custom InstallShield ICEs (ISICEs) that are available for the some of the InstallShield installation and merge module validation suites. The ISICEs help you validate your package against best practices for Windows-based installations.



Windows Logo *• Validating your project may help you identify whether your installation meets installation requirements for Microsoft's Windows logo program. Therefore, if you are interested in being able to use the Windows logo, it is recommended that you use the* ***InstallShield Validation Suite for Windows 8*** *and the* ***Full MSI Validation Suite*** *to validate your installation package. If you create a merge module in InstallShield, use the* ***InstallShield Merge Module Validation Suite for Windows 8*** *and the* ***Merge Module Validation Suite*** *to validate your merge module. For more details, see Requirements for the Windows Logo Program.*

To learn more about the Windows logo program, visit MSDN.



Edition *• The Premier Edition of InstallShield includes the following sets of validation suites:*

The InstallShield Premier Edition also includes the following sets of validators:

- ***InstallShield Virtualization Suitability Suite****—ISVICEs validators in this suite help you to determine how ready your products are for virtualization by checking suitability for Microsoft App-V 4.x, Microsoft App-V 5.x, Microsoft Server App-V, VMware ThinApp, and Citrix XenApp. The validation suites can enable you to make more informed decisions about how you would build your product if you are considering offering your customers a virtualized version.*
- ***InstallShield Best Practice Suite****—ISBP validators in this suite alert you if your installation violates best-practice guidelines.*
- ***InstallShield UWP App Suitability Suite****—ISUWP validators in this suite scan an .msi package for signs of items that are unsuitable for the UWP app package format and provides a report that indicates applicability to the known UWP app variants (Universal App, Desktop Bridge, Windows Store, WSA, and Nano Server).*

InstallShield also provides an engine for upgrade and patching validation. You can access this through the Upgrade Validation Wizard.

Validating an Installation Package or Merge Module

There are two ways to validate an installation package or a merge module from within InstallShield:

- You can configure InstallShield to validate the installation package or merge module every time that you build a release. This is disabled by default. For more information, see Specifying Whether Validation Should Be Performed at Build Time.

  If you perform validation at build time, you can specify multiple validation suites.

- You can run validation on demand for a built release. To learn how, see Validating an Installation Package or Merge Module on Demand.

  If you perform validation on demand, you can specify only one validation suite at a time.



Tip *• As an alternative, you can validate the installation package or merge module after you build it from the command line by using* **ISCmdBld.exe***. For more information, see ISCmdBld.exe.*



Project *• The only way to validate a package for an MSI Database project or an MSM Database project is to perform validation on demand.*

***Note** • If you want to see validation warnings that apply to your installation or merge module, you must perform validation on demand; this type of validation message is not available if you perform validation at build time. InstallShield reports the other types of validation messages—errors and failures—during both validation methods. To learn more about the different types of validation messages, see Viewing Validation Results.*

Specifying Whether Validation Should Be Performed at Build Time

InstallShield enables you to specify whether installation packages and merge modules should be validated each time that a release is successfully built. InstallShield also lets you specify one or more validation suites that should be used for validation.

Task ***To configure validation settings:***

1. On the **Tools** menu, click **Options**. The **Options** dialog box opens.
2. Click the **Validation** tab.
3. Select the check boxes for the types of validation that you would like InstallShield to perform at build time. If you prefer to validate on demand, and not at build time, clear the check boxes.
4. If you select the **Perform validation using** check box or the **Perform Merge Module validation using** check box, select one or more types of validation that should be performed.

   To add a new or custom validator (.cub file), click the **Browse** button and select it.

***Note** • If you want to see validation warnings that apply to your installation or merge module, you must perform validation on demand; this type of validation message is not available if you perform validation at build time. InstallShield reports the other types of validation messages—errors and failures—during both validation methods.*

Specifying Which ICEs, ISICEs, ISVICEs and ISBPs Should Be Run During Validation

InstallShield lets you customize the list of ICEs, ISICEs, ISVICEs, ISUWPs, and ISBPs that should be used for installation package validation and merge module validation.

Task ***To specify which ICEs, ISICEs, ISVICEs, ISUWPs, and ISBPs should be run during validation:***

1. On the **Tools** menu, click **Options**. The **Options** dialog box opens.
2. Click the **Validation** tab.
3. Click the **Customize** button. The **Customize Validation Settings** dialog box opens.
4. In the **Select CUB file to customize** list, click the suite that you want to customize.

5. In the list of ICEs, select the check box for each of the validators that should be used to evaluate the installation package or merge module. Clear the check box for each of the validators that should not be used for the validation.

   To configure multiple consecutive validators, select the first file, press and hold SHIFT, and select the last file. Then either select or clear the check box of one of the selected validators.

6. Click **OK**.

If you customize the list of ICEs, ISICEs, ISVICEs, ISUWPs, and ISBPs for a validation suite, anytime that validation is performed—whether it occurs at build time or on demand—only the selected ICEs, ISICEs, ISVICEs, ISUWPs, and ISBPs are used.

Validating an Installation Package or Merge Module on Demand

InstallShield enables you to validate an installation package or merge module separately from the build process. Doing so is especially helpful if you configured InstallShield so that validation is not performed as part of every successful build, but you do need to run it at some point to test your product for Windows logo compliance or for best-practice compliance.

Task ***To validate your release on demand:***

1. Complete a successful build of a release.
2. On the **Build** menu, point to **Validate**, and then click the validation type that you want to run.

Viewing Validation Results

InstallShield displays the results of the validation scan on the Validations tab of the Output window. In addition, InstallShield saves the results to an XML document in a **ValidationFiles** folder within the release folder. You can view this file either by navigating to your build directory, or by navigating to the Releases view and selecting the Validations folder under your release.

Validation Messages

Validation messages are broken down into three categories:

- **Errors**—Describe problems with your database, such as having duplicate component GUIDs.
- **Warnings**—Describe problems in your database that may occur in certain circumstances.
- **Failures**—Occur when your database has severe enough problems that the validation tool might not be able to run.

If the scan results for your project include validation messages, the messages and associated codes are also listed on the Tasks tab of the Output window.

Note *• If you want to see validation warnings that apply to your installation or merge module, you must perform validation on demand; this type of validation message is not available if you perform validation at build time.*

InstallShield reports the other types of validation messages—errors and failures—during both validation methods. To learn more about these two validation methods, see Validating an Installation Package or Merge Module.



***Tip** • You can click a validation code on the Tasks tab of the Output window to see the corresponding Knowledge Base article or HelpNet topic.*

In addition, you can click a validation message on the Tasks tab to jump to the area of the Direct Editor that corresponds to the validation message. This functionality is available for: ICEs, ISICEs, ISVICEs, ISUWPs, and ISBPs.

ICEs

The validation tool checks your project for compliance with each of the internal consistency evaluators, or ICEs. These ICEs are those used by **Msival2.exe** (part of the Microsoft Windows Installer Platform SDK) to validate installation and merge module packages for Windows logo compliance.

If your installation project or merge module project fails one or more of these ICEs, InstallShield reports the specific ICE rules that were violated and offers additional information to help you troubleshoot the problem.

To learn about specific ICEs, see ICE Reference in the Windows Installer Help Library.

ISICEs

The InstallShield validation suites consist of a number of InstallShield ICEs (ISICEs) that help you check your project for compliance with installation requirements of the Windows logo program.

The following table lists the ISICEs that are available in InstallShield.

Table 4-18 • ISICEs

ISICE	Project Type	Description
ISICE01	Basic MSI, InstallScript MSI, MSI Database	Validates that the application is installed to ProgramFiles or the user's AppData folder by default.
ISICE02	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that all executable files included with the installation or merge module are digitally signed.
ISICE03	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that there are no nested-installation custom actions (type 7, 23, and 39).
ISICE04	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that there are no custom columns added to standard .msi tables.

Table 4-18 • ISICEs (cont.)

ISICE	Project Type	Description
ISICE05	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that there are no properties or custom tables that begin with MSI (case-insensitive).
ISICE06	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that the installation does not include any files protected by Windows Resource Protection.
ISICE07	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that MSI components are authored for proper installation and uninstallation.
ISICE08	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that deferred custom actions do not have the impersonate bit set.
ISICE09	Basic MSI, InstallScript MSI, MSI Database	Validates that the installation includes an MsiPatchCertificate table entry.
ISICE10	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that each custom action has a path specified for the Help File Path setting in the Custom Actions view.
ISICE11	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that .exe files have embedded manifests with required information.
ISICE12	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that no protected registry keys are modified.
ISICE13	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that no .hlp files or WinHelp run-time files are included in the installation. This ICE is now ISBP17.

Table 4-18 • ISICEs (cont.)

ISICE	Project Type	Description
ISICE14	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that no obsolete APIs are imported. This ICE is now ISBP18.
ISICE15	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that no deprecated APIs are imported. This ICE is now ISBP19.
ISICE16	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that no 16-bit components are distributed in an installation that targets 64-bit systems.
ISICE17	Basic MSI, InstallScript MSI, Merge Module, MSI Database	Validates that the installation or merge module does not force a reboot.
ISICE18	Basic MSI, MSI Database	Validates that the MsiRMFilesInUse dialog is included.
ISICE19	Basic MSI, InstallScript MSI, MSI Database	Validates properties related to upgrades, that the Upgrade table is present, and that the installation prevents downgrades.
ISICE20	Basic MSI, InstallScript MSI, MSI Database	Validates that no machine-wide settings are installed by a non-privileged installation.
ISICE21	Basic MSI, InstallScript MSI, MSI Database	Validates that the app does not depend on the VB6 runtime.
ISICE22	Basic MSI, InstallScript MSI, MSI Database	Validates that the app does not start automatically on system startup.
ISICE23	Basic MSI, InstallScript MSI, MSI Database	Validates that the app sets strong and appropriate ACLs.
ISICE24	Basic MSI, InstallScript MSI, MSI Database	Validates that the installation does not install 16-bit components.

Table 4-18 • ISICEs (cont.)

ISICE	Project Type	Description
ISICE25	Basic MSI, InstallScript MSI, MSI Database	Validates that the installation does not install any files to the Windows directory or its subdirectories.
ISICE26	Basic MSI, InstallScript MSI, MSI Database	Validates that applications support Windows security features by controlling permissions appropriately.

Tip • *In some cases, the following validation error message may be displayed for an ISICE:*

`File [1] in Component [2] could not be found. All tests against this file's contents may be invalid.`

This error occurs if the specified file is missing, and the associated ISICE could not be verified for the file. For example, if an executable file is missing, ISICE11 cannot verify whether the file has an embedded manifest.

If the aforementioned validation error message is displayed, resolve the missing file issue, and then rerun validation for your project.

ISICE01

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
The application should be installed to either ProgramFiles or AppData folder by default. The current
    default location is [1].
```

[1] is the default installation location for the product.

Description

Users should be able to install products where they need them, and they should have a consistent experience with where files are stored by default.

ISICE01 verifies that the default destination location for your product is the Program Files folder or the Application Data folder. If a different location is used, this error message is displayed during validation.

Corrective Action

Change the default location. For more information, see Setting the Default Product Destination Folder (INSTALLDIR).

ISICE02



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Message (Error)

```
File [1] in Component [2] is not digitally signed. All [3] files distributed to Windows Vista and later
    are required to be signed.
```

[1] is the name of an executable file, [2] is the name of the component that contains that file, and [3] is the type of file that must be signed.

Description

ISICE02 verifies that all executable files in your installation have been digitally signed. This includes files with the following extensions: exe, dll, ocx, sys, cpl, drv, and scr. If an executable file in your installation has not been digitally signed, this error message is displayed during validation.

Corrective Action

Ensure that all executable files in your installation have been digitally signed. For more information, see Digital Signing and Security.

ISICE03



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Message (Error)

```
Nested custom action [1] of type [2] is not allowed.
```

[1] is the name of a custom action in your project, and [2] is the Windows Installer type number of custom action.

Description

ISICE03 verifies that your installation does not include any nested-installation custom actions. Nested installations is a deprecated feature of the Windows Installer. Applications installed with nested installations sometimes fail because they are difficult for end users to service correctly. Microsoft Corporation recommends that you avoid using nested installations and nested-installation custom actions to install products that are intended to be released to the public. To learn more, see Concurrent Installations in the Windows Installer help library.

Corrective Action

To resolve this error, remove the nested-installation custom action from your project. The MSI Type Number value in the Custom Actions and Sequences view (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or the Custom Actions view (in Merge Module and MSM Database projects) for nested-installation custom actions is 7, 23, or 39.

As an alternative to using a nested installation, you may want to create a create an InstallShield prerequisite, and then add that InstallShield prerequisite to your installation project. For more information, see Defining InstallShield Prerequisites.

ISICE04



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Message (Error)

```
The standard MSI table [1] does not match the MSI schema defined in schema.msi.
```

[1] is the name of a table in your project.

Description

ISICE04 verifies that your installation does not add custom table columns to standard tables. Adding columns to standard tables is reserved for future versions of Windows Installer.

Corrective Action

To resolve this error, remove any custom table columns that were added to the table that is mentioned in the error message. You can do this by exporting the table from the Direct Editor, editing the table in a text editor, and importing the revised table.

ISICE05



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Messages

Message 1 (Error)

```
MSI property [1] begins with reserved characters. MSI property names cannot start with "MSI" (case-
    insensitive).
```

[1] is the name of a property in your project.

Message 2 (Error)

```
Table [1] begins with reserved characters. Custom table names cannot start with "MSI" (case-
    insensitive).
```

[1] is the name of a table in your project.

Description

ISICE05 verifies that your installation does not include any custom properties or custom tables whose names begin with **MSI** in any combination of uppercase and lowercase letters. The **MSI** prefix is reserved for future use in new standard properties and tables.

Corrective Action

To resolve the property-related error, use the Property Manager to rename the custom property that is mentioned in the error message.

To resolve the table-related error, use the Direct Editor to export the table, use a text editor to edit the table name in the exported .idt file, and use the Direct Editor to import the newly renamed table into the Direct Editor. Then delete the original custom table.

ISICE06



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Messages

Message 1 (Error)

```
File [1] in Component [2] is a Windows Protected File. Protected files may not be distributed to Windows
    Vista and later.
```

[1] is the name of a file in your project, and [2] is the name of the component that contains that file.

Message 2 (Warning)

```
File [1] in Component [2] has the same file name as a Windows Protected File, but appears to be
    distributed to a different path.
```

[1] is the name of a file in your project, and [2] is the name of the component that contains that file.

Description

ISICE06 verifies that your installation does not install files that are protected by Windows Resource Protection (WRP). WRP is designed to ensure that protected system resources are updated on Windows Vista and later machines only through Microsoft-approved installation or update mechanisms.

If your installation installs a file that has the same name as a Windows Protected File, but the file is not installed to the same location as the Windows Protected File, warning message 2 is displayed.

Corrective Action

To resolve the validation error, remove the file that is mentioned in the error message from your project.

If you encounter the validation warning, determine if the file that is mentioned in the message is a protected file:

- If the file is not a protected file but it has the same name as a protected file, you can ignore this warning.
- If the file is a protected file, remove it from your project to resolve this warning.

If your product requires newer versions of system components, the components must be updated on target machines by using a Microsoft service pack or a Microsoft-approved installation package that contains the system component. System components should not be repackaged.

ISICE07



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Messages

Message 1 (Error)

```
Component.ComponentId for Component [1] is null. All components must have a valid ComponentId for proper
    installation and uninstallation. If this is left null, justification must be documented.
```

[1] is the name of a component in your project that does not have a GUID specified in the Component Code setting in the Components view.

Message 2 (Error)

```
Component [1] contains COM data, but it has more than 1 file. Each COM component should contain only 1
    COM file.
```

[1] is the name of a component in your project that contains more than one COM server.

Message 3 (Error)

```
Component [1] has more than one file as the target of a Desktop or Start Menu shortcut.
```

[1] is the name of a component in your project that has more than one file specified as the target of a shortcut.

Message 4 (Error)

```
Component [1] is associated with multiple features and is referenced in the following tables: [2]. These
    references require the component to be associated with exactly 1 feature.
```

[1] is the name of a component in your project that is associated with more than one feature and that has references in any of the following tables: **Class**, **Extension**, **MsiAssembly**, **PublishComponent**, **TypeLib**. [2] is a comma-delimited list of tables that contain references to the component.

Description

ISICE07 verifies that the components in your installation adhere to component rules, which help to ensure that the installation or uninstallation of one product does not harm any other product on a target system. These rules also help to ensure that Windows Installer correctly removes all resources that are connected with a product that is being uninstalled.

Corrective Action

To resolve error 1, open the Components view, select the component that is mentioned in the error message, and click the Component Code setting. In the help pane that is displayed in the lower-right pane, click Generate GUID. If you have a valid reason for leaving the Component Code setting blank, you can document it in the application that you submit for the Windows logo program.

To resolve error 2, move any COM servers to separate components so that each COM server is the key path of its component.

To resolve error 3, use the Shortcuts view to delete all but one of the shortcuts that are associated with the component mentioned in the error message.

To resolve error 4, use the Setup Design view or the Features view to remove the component from all but one feature.

ISICE08



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Message (Error)

```
Custom Action [1] uses impersonation. Impersonation should not be used when running setups on Windows
    Vista or later.
```

[1] is the name of a custom action in your project.

Description

ISICE08 verifies that your installation does not contain any custom actions that use impersonation. If you select one of the Terminal Server Aware options for the type of in-script execution setting (in the Custom Actions and Sequences view, in the Custom Actions view, or on the Respond Options panel of the Custom Action Wizard), the custom action impersonates the end user during per-machine installations on terminal server machines.

Corrective Action

To resolve this validation error, open the Custom Actions and Sequences view (or the Custom Actions view) and click the custom action that is mentioned in the error message. Change the value of the In-Script Execution setting to one of the options that does not include Terminal Server Aware.

ISICE09



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
An MsiPatchCertificate table entry is required for User Account Control (UAC) patching.
```

Description

ISICE09 verifies that your installation includes an **MsiPatchCertificate** table entry. In a base installation, this table provides the signer certificate that will be used to verify the digital signature of subsequent patches when they are applied by a non-administrator or an administrator who is not using elevated privileges.

Corrective Action

To resolve this validation error, add the **MsiPatchCertificate** table to your project. To learn how, see Preparing Installations for Non-Administrator Patches.

ISICE10



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *Merge Module*
- *MSI Database*

Message (Error)

```
Custom action [1] of type [2] is not documented in table [3].
```

[1] is the name of a custom action in your project, and [2] is its Windows Installer type number. [3] is the name of the table that is missing an entry for the custom action that is listed.

Description

The intended behavior of each custom action must be documented for the Windows logo program. This is especially helpful if system administrators deploy your product to enterprise environments; they sometimes need to know what the custom actions do.

ISICE10 verifies that each custom action in your installation is documented by validating that each entry in the **CustomAction** table has a corresponding **ISCustomActionReference** table entry.

Corrective Action

To resolve this validation error, open the Custom Actions and Sequences view (or the Custom Actions view), select the custom action that is mentioned in the error message, and use the Help File Path setting to specify a path to the document that describes the behavior of the custom action. When you specify a value in the Help File Path setting, InstallShield adds a row for that custom action in the **ISCustomActionReference** table if one has not already been created.

In addition, configure the product configuration for the release so that InstallShield streams the contents of each of the custom action help files into the .msi file at build time. For more information, see the description of the Include Custom Action Help setting on the General tab for a product configuration in the Releases view.

Note that if the custom action is a merge module that is consumed in your installation project, specify the path in the Custom Actions view of the merge module project and then rebuild the merge module.

ISICE11



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Messages

Message 1 (Error)

```
Exe [1] in component [2] lacks a manifest.
```

[1] is the name of an executable file in your project that does not have a manifest, and [2] is the name of the component that contains the executable file.

Message 2 (Error)

```
Exe [1] in component [2] lacks a requestedExecutionLevel level setting in its manifest.
```

[1] is the name of an executable file in your project that has a manifest that was possibly created for Windows XP but that was not updated for Windows Vista and later. The manifest does not define the execution level for the executable file. [2] is the name of the component that contains the executable file.

Message 3 (Error)

```
Exe [1] in component [2] lacks a requestedExecutionLevel uiAccess setting in its manifest.
```

[1] is the name of an executable file in your project that has a manifest that was possibly created for Windows XP but that was not updated for Windows Vista and later. The manifest does not define the UI accessibility value for the executable file. [2] is the name of the component that contains the executable file.

Message 4 (Warning)

```
Exe [1] in component [2] is marked to require elevated privileges (highestAvailable) which requires a
    waiver from Microsoft.
```

[1] is the name of an executable file in your project that has a manifest that specifies that elevated privileges are required, and [2] is the name of the component that contains the executable file.

Message 5 (Warning)

```
Exe [1] in component [2] is marked to require elevated privileges (requireAdministrator) which requires
    a waiver from Microsoft.
```

[1] is the name of an executable file in your project that has a manifest that specifies that elevated privileges are required, and [2] is the name of the component that contains the executable file.

Message 6 (Warning)

```
Exe [1] in component [2] is marked to require uiAccess which requires a waiver from Microsoft.
```

[1] is the name of an executable file in your project that has a manifest that specifies that the executable file is allowed to bypass UI protection levels in order to use higher privileges to pass information to other windows on the desktop; that is, the value of uiAccess is set to true. [2] is the name of the component that contains the executable file.

Description

ISICE11 verifies that every .exe file in your installation has an embedded manifest that defines its execution level. The execution level is defined in the manifest as follows:

```
<requestedExecutionLevel
  level="asInvoker"
  uiAccess="false"/>
```

Other valid values for the level attribute are highestAvailable and requireAdministrator. Note that if you set the level value to highestAvailable or requireAdministrator, you must apply for a waiver from Microsoft to obtain Windows logo certification.

The uiAccess attribute indicates whether the executable file is allowed to bypass UI protection levels in order to use higher privileges for passing information to other windows on the desktop, such as on-screen keyboards. This attribute should be set to true only for UI accessibility applications. Note that if you set the UI accessibility value to true, you must apply for a waiver from Microsoft to obtain Windows logo certification.

Corrective Action

To resolve the validation errors (messages 1 through 3), replace the executable file in your installation with one that has an embedded manifest whose level and uiAccess values are set appropriately.

The validation warnings (messages 4 through 6) are displayed to inform you that if you want to have your product qualify for the Windows logo program but you keep the requestedExecutionLevel element as it is currently defined in the executable file's manifest, you may need to obtain a waiver from Microsoft.

ISICE12



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Message (Error)

```
Registry key [1] in Component [2] is a protected registry key. Protected registry keys may not be
    modified on Windows Vista and later.
```

[1] is a registry key that only the Trusted Installer group can modify on Windows Vista or later systems, and [2] is the name of the component that contains that registry key.

Description

ISICE12 verifies that the installation does not attempt to modify registry keys that are protected by Windows Resource Protection (WRP) on Windows Vista and later systems. Registry keys that can be modified by only the Trusted Installer group are protected by WRP.

Corrective Action

To resolve this validation error, use the Registry view to remove the protected registry key from your project, or move it to a part of the registry that is not protected by WRP.

ISICE16



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *Merge Module*
- *MSI Database*

Message (Error)

```
File [1] in Component [2] is a 16-bit file. Installations that target 64-bit systems may not include 16-bit files.
```

[1] is the name of a 16-bit file in your project, and [2] is the name of the component that contains the file.

Description

If your installation targets 64-bit platforms, ISICE16 verifies that it does not contain any 16-bit files, since 64-bit platforms do not support 16-bit files.

Corrective Action

To resolve this validation error, replace the 16-bit file with a 64-bit or 32-bit file.

ISICE17



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

Messages

Message 1 (Error)

```
Sequence [1] includes the [2] action. Use of the [2] action requires a waiver from Microsoft.
```

[2] is the name of an action in your project, and [1] is the name of the sequence that contains that action.

Message 2 (Error)

```
Control [1] on Dialog [2] runs the [3] action. Use of the [3] action requires a waiver from Microsoft.
```

[3] is the name of an action in your project. [2] is the name of the control that includes an event that launches that action. [2] is the name of the dialog that has the control.

Description

ISICE17 verifies that the installation does not use an action to force a reboot. Currently, the only action that is validated is ForceReboot.

Corrective Action

The errors are displayed to inform you that if you want to have your product qualify for the Windows logo program but you keep the specified action in your installation, you may need to obtain a waiver from Microsoft.

To resolve these validation errors, remove the action from sequence or dialog that is mentioned in the message.

Ensure that your project includes the MsiRMFilesInUse dialog to minimize system reboots, and that your application has been properly instrumented to use the Restart Manager API. For more information, see Minimizing Reboots on Windows Vista and Later Systems.

ISICE18



Project • *This information applies to the following project types:*

- *Basic MSI*
- *MSI Database*

Messages

Message 1 (Error)

```
Dialog [1] is not present in the Dialog table.
```

[1] is the name of a dialog that should be in your project.

Message 2 (Error)

```
The Title of Dialog [1] does not include [2] or [ProductName].
```

[1] is the name of a dialog in your project, and [2] is the value of the [ProductName] variable.

Description

ISICE18 verifies that the installation includes the MsiRMFilesInUse dialog, and that the dialog's title bar contains the name of the product.

Corrective Action

To resolve error 1, ensure that the MsiRMFilesInUse dialog is in your project. All Basic MSI projects include this dialog by default. For more information about this dialog, see Minimizing Reboots on Windows Vista and Later Systems.

To add the dialog back to a Basic MSI project, you can open another Basic MSI project that contains this dialog, or create a new Basic MSI project. Then export the dialog to the project that needs it. For more information, see Exporting Dialogs to Other Projects.

To resolve error 2, add the product name or the [ProductName] property to the Caption property for the MsiRMFilesInUse dialog.

ISICE19



Project • *This information applies to the following project types:*

- *Basic MSI*

- *InstallScript MSI*
- *MSI Database*

Messages

Message 1 (Error)

```
The table [1] is not present in the installation package.
```

[1] is the name of a table that must be in the package.

Message 2 (Error)

```
UpgradeCode [1] does not detect and prevent downgrades with a properly scheduled Type 19 Custom Action;
    the package must prevent downgrades.
```

[1] is the value of the UpgradeCode property.

Message 3 (Error)

```
The property [1] is not listed in SecureCustomProperties; this can prevent it from being used correctly
    for downgrade prevention.
```

[1] is the name of a property.

Message 4 (Error)

```
The property [1] must be a valid GUID; [2] is not a valid GUID.
```

[1] is the name of a property whose value should be a valid GUID. [2] is the invalid value that is currently used for that property.

Message 5 (Error)

```
Column [1] of table Upgrade must hold a valid GUID; [2] is not a valid GUID.
```

[1] is the name of a column in the **Upgrade** table, and [2] is the value of an entry in the specified column.

Message 6 (Error)

```
The property [1] must hold a valid numeric version in the format Major.Minor.Build; [2] is not a version
    in this format.
```

[1] is the name of a property whose value should be a version number in the Major.Minor.Build format. [2] is the invalid value that is currently used for that property.

Message 7 (Error)

```
Column [1] of table Upgrade must hold a valid numeric version in the format Major.Minor.Build; [2] is not
    a version in this format.
```

[1] is the name of a column in the **Upgrade** table, and [2] is the value of an entry in the specified column.

Message 8 (Error)

```
The property [1] must not be null or empty.
```

[1] is the name of a property that is null or empty.

Message 9 (Error)

```
Column [1] of table Upgrade must not be null or empty.
```

[1] is the name of a column in the **Upgrade** table.

Message 10 (Error)

```
The property [1] must be provided; [2] is the default value.
```

[1] is the name of a property in your project that has not been customized. [2] is the default value that InstallShield uses for that property.

Message 11 (Error)

```
The property [1] must hold a valid numeric LANGID; [2] is not a numeric LANGID.
```

[1] is the name of a property whose value should be a valid language identifier. [2] is the invalid value that is currently used for that property.

Description

ISICE19 validates properties related to upgrades and that the **Upgrade** table is present. ISICE19 also helps you determine whether the installation prevents an earlier package from installing over a new package.

Corrective Action

To resolve errors 1 and 2, ensure that the Upgrades view contains a major upgrade item, that the major upgrade item is properly configured to prevent the current version of your product from being installed over a future version, and that your project includes a properly configured and scheduled type 19 custom action. For detailed instructions, see Preventing the Current Installation from Overwriting a Future Major Version of the Same Product. For more information, see Preventing an Old Package from Installing Over a Newer Version in the Windows Installer Help Library.

To resolve error 3, add the specified property to the value of the **SecureCustomProperties** property. You can use the Property Manager view to set a value for a property. To add multiple values, separate each with a semicolon. For more information, see SecureCustomProperties Property in the Windows Installer Help Library.

To resolve errors 4 and 5, replace the existing value with a valid GUID. To change the value, open the General Information view. Then click the setting that is listed in the error message. Enter a valid GUID. Note that if you want to use a new, unique GUID, you can click the **Generate a new GUID** button in the setting. For more information, see GUIDs.

To resolve errors 6 and 7, replace the specified value with a valid version number; you can click the error message in the Output window to move to the location in the Direct Editor that has the invalid value. The version number must contain only numbers, and it must be in the format **aaa.bbb.ccccc**, where *aaa* represents the major version number, *bbb* represents the minor version number, and *ccccc* represents the build number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* is 65,535. For more information, see Specifying the Product Version.

To resolve error 8, enter a value for the specified property. You can use the Property Manager view to set a value for a property.

To resolve error 9, enter a value for the specified column. You can use the Direct Editor view to set the value; you can click the error message in the Output window to move to the location in the Direct Editor that needs to be revised. To learn more about any of the Windows Installer tables, see Database Tables in the Windows Installer Help Library.

To resolve error 10, replace the default value of the specified property with the appropriate value. You can use the Property Manager view to change the value.

To resolve error 11, replace the existing value of the specified property with a valid language identifier.

ISICE20



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Messages

Message 1 (Error)

```
Row [1] of Table [2]: installing, modifying, or deleting a registry key in root [3] requires elevated
    privileges, but this package does not request elevated privileges.
```

[1] is the name of the row that contains information about a registry change in your project. [2] is the name of the table that contains that registry-related data. [3] is the predefined root key for the registry data, as listed in the Root column for the specified row.

Message 2 (Error)

```
Row [1] of Table [2]: installing, modifying, or deleting a file in directory [3] requires elevated
    privileges, but this package does not request elevated privileges.
```

[1] is the name of a row that contains information about a directory that end users cannot access without elevated privileges. [2] is the name of the table that contains that directory data. [3] is the name of the directory.

Message 3 (Error)

```
Environment variable [1] is a system environment variable, but this package does not request elevated
    privileges.
```

[1] is the name of a system environment variable that is included in your project.

Description

If the Require Administrative Privileges setting in the General Information view is set to No for your project, ISICE20 verifies that your installation does not attempt to perform tasks that require administrative privileges, such as writing to system registry or folder locations.

Corrective Action

To resolve ISICE20 errors, do one of the following:

- Select Yes for the Require Administrative Privileges setting in the General Information view. For more information, see Entering Summary Information Stream Data.

- Change your project so that it does not install, modify, or delete data in system locations. For example, if your project adds a registry key to HKEY_LOCAL_MACHINE, use the Registry view to move that registry key to HKEY_CURRENT_USER. If your project adds or modifies a system environment variable, use the Environment Variables view to change the Type setting of that environment variable from System to User.

ISICE21



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
File [1] in Component [2] depends on the VB6 runtime.
```

[1] is the name of a file that requires the Visual Basic 6 runtime. [2] is the name of the component that contains that file.

Description

ISICE21 verifies that your product does not depend on the Visual Basic 6 runtime.

Corrective Action

To resolve this validation error, consider the following options:

- If the file is in a component that your product no longer uses, remove the file and its component from your project. Consider using a newer language such as Visual Basic .NET for the creation of your file.
- Remove the file and its component from your project and replace them with new versions that do not rely on the Visual Basic 6 runtime.

If neither of those options are feasible for your product, your installation will not qualify for the Windows logo program.

ISICE22



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Messages

Message 1 (Error)

`Shortcut [1] in Component [2] targets the startup folder.`

[1] is the name of a shortcut in your project, and [2] is the name of the component that contains that shortcut.

Message 2 (Error)

`Registry key [1] in Component [2] sets a Run key.`

[1] is a registry key in your project, and [2] is the name of the component that contains that shortcut.

Description

ISICE22 verifies that your product does not start automatically when the target system starts.

Error message 1 occurs if your installation creates a shortcut in the Startup folder on the Start Menu.

Error message 2 occurs if your installation sets a Run key in locations of the registry such as the following ones:

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion
- HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows\CurrentVersion
- HKEY_CURRENT_USER\Software\Wow6432Node\Microsoft\Windows\CurrentVersion

Corrective Action

To resolve this validation error, remove the shortcut or registry key and—if appropriate—the component that contains it from your project.

ISICE23



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

`LockObject [1] in Component [2] sets insecure permissions.`

[1] is the name of a file, directory, or registry key for which permissions are configured. [2] is the name of the component that contains that file, directory, or registry key.

Description

ISICE23 verifies that your installation does not expose a target system to security risks through the configuration of inappropriate permissions for files, directories, and registry keys.

For example, if your installation assigns write permission to the Everyone user for a particular folder that your installation installs, ISICE23 triggers an error to alert you to the security vulnerability to which your installation exposes target systems.

Corrective Action

To resolve this validation error, use strong and appropriate permissions for the file, directory, or registry key that is mentioned in the error. For more information, see:

- Configuring Permissions for Files and Folders
- Configuring Permissions for Registry Keys

ISICE24



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
File [1] in Component [2] is a 16-bit file. Installations that target Windows 8 systems may not include
    16-bit files.
```

[1] is the name of a 16-bit file in the installation. [2] is the name of the component that contains that file.

Description

ISICE24 verifies that the installation does not contain any 16-bit files; 64-bit versions of Windows do not support 16-bit files.

Corrective Action

To resolve this validation error, replace the 16-bit file with a 64-bit or 32-bit file.

ISICE25



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
The destination of Component [1] is the Windows folder or one of its subfolders.
```

[1] is the name of a component.

Description

ISICE25 verifies that the installation does not write directly to the Windows folder or any of its subfolders.

Corrective Action

To resolve this validation error, do one of the following:

- Change the destination of the component.
- If the component is installing a font, ensure that the font file is included in the project according to best practices. To learn how, see Installing Fonts.
- If the component is installing a device driver, ensure that the driver is included in the project according to best practices. To learn how, see Configuring Device Driver Settings.

ISICE26



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
File [1] in Component [2] does not have the Option Header bit [3] set.
```

[1] is the name of a file, and [2]is the name of the component that contains the file. [3] is the parameter (SafeSEH, NXCOMPAT, or DYNAMICBASE) that was not used when compiling the file.

Description

ISICE26 verifies that the installation supports Windows security features. It checks the PE header of the executable files that the installation installs to verify that they were compiled using the following command-line parameters:

- /SafeSEH—to ensure safe exceptions handling
- /NXCOMPAT—to prevent data execution
- /DYNAMICBASE—for address space layout randomization (ASLR)

A single file can trigger multiple ISICE26 errors, one for each of the aforementioned parameters.

Corrective Action

To resolve this validation error, review build the options for your product's source code, and edit them as needed. In Visual Studio, these are configured on the Advanced property page for Linker settings.

InstallShield Virtualization Suitability Suite



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

InstallShield includes a set of validators called the InstallShield Virtualization Suitability Suite. The InstallShield Virtualization Suitability validators (ISVICEs) in this suite help you check your project's suitability for various virtualization technologies.

The following table lists the ISVICEs that are available in InstallShield.

Table 4-19 • ISVICEs

ISICE	Project Type	Technology	Description
ISVICE01	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, Server App-V, ThinApp, XenApp	Checks for the presence of files that may normally be part of the Windows operating system.
ISVICE02	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, Server App-V, ThinApp, XenApp	Checks for the presence of device driver files.
ISVICE03	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, Server App-V, ThinApp, XenApp	Checks for the presence of ClickOnce deployment files.
ISVICE04	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, ThinApp, XenApp	Checks for the presence of files that are part of ASP.NET/IIS applications.
ISVICE05	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, ThinApp, XenApp	Checks for the presence of files that are part of WMI providers.

Table 4-19 • ISVICEs (cont.)

ISICE	Project Type	Technology	Description
ISVICE06	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, Server App-V, ThinApp, XenApp	Checks for the presence of files that are part of J2EE application servers.
ISVICE07	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, Server App-V, ThinApp, XenApp	Checks for the presence of files that are part of applications that may not work when virtualized.
ISVICE08	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, Server App-V, ThinApp, XenApp	Checks for the presence of files that are part of applications that are known to fail when virtualized.
ISVICE09	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, ThinApp, XenApp	Checks for the presence of InstallShield IIS support indicating that an ASP.NET/IIS application is installed.
ISVICE10	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, Server App-V, ThinApp, XenApp	Checks for the presence of drivers installed through the **MsiDriverPackages** table.
ISVICE11	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, ThinApp, XenApp	Checks the package to ensure that it contains at least one shortcut.
ISVICE12	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, Server App-V, ThinApp, XenApp	Checks for the presence of shell extensions.
ISVICE13	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, Server App-V, ThinApp, XenApp	Checks for the presence of URL protocol handlers.

Table 4-19 • ISVICEs (cont.)

ISICE	Project Type	Technology	Description
ISVICE14	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, Server App-V, ThinApp, XenApp	Checks for the presence of support for the Default Programs list.
ISVICE15	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, ThinApp, XenApp	Checks for the presence of boot start services.
ISVICE16	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, Server App-V, XenApp	Checks to ensure that the total installed file size is less than 4 GB.
ISVICE17	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, ThinApp, XenApp	Checks for the usage of the Windows Installer COM+ tables.
ISVICE18	Basic MSI, InstallScript MSI, MSI Database	App-V 4.x, App-V 5.x, ThinApp, XenApp	Checks for the presence of COM DLL surrogate files.
ISVICE19	Basic MSI, InstallScript MSI, MSI Database	ThinApp, XenApp	Checks to see if this is a 64-bit package. ThinApp and XenApp do not support 64-bit applications.

ISVICE01

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE01 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- Server App-V
- ThinApp
- XenApp

Message (Error)

```
The file [1] appears to be a file that is closely integrated with the OS. The files that make up
    applications like Internet Explorer or Windows Media Player, or frameworks like the .NET Framework,
    do not make good candidates for virtualization. These files should instead be installed locally on
    the machine.
```

[1] is the name of a file in the project.

Description

ISVICE01 checks for the presence of files that may normally be part of the Windows operating system. If the release contains one or more of those type of files, this error message is displayed during validation.

Corrective Action

If your end users may be interested in a virtualized version of your product, and if your InstallShield project contains the .NET Framework or other items that are closely integrated with Windows, you may want to remove those items from your project. You could document the technologies that are required for your product and instruct your end users to install them locally on their machines before using a virtualized version of your product.

ISVICE02



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE02 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- Server App-V
- ThinApp
- XenApp

Message (Error)

```
The file [1] appears to be a device driver. System-level drivers such as print drivers or USB device
    drivers do not work from a virtualized environment. It may be possible to extract this driver such
    that it can be installed locally on the machine and allow the rest of the package to be virtualized.
```

[1] is the name of a file in the project.

Description

ISVICE02 checks for the presence of device driver files. System-level drives such as print drivers or USB device drivers do not work from a virtualized environment.

Corrective Action

If your end users may be interested in a virtualized version of your product, and if your InstallShield project contains a device driver, consider removing the device driver component from your project, and creating a separate installation for the driver. End users can then install the driver locally to their machines before using a virtualized version of your product.

ISVICE03



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE03 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- Server App-V
- ThinApp
- XenApp

Message (Error)

```
The file [1] appears to be a ClickOnce deployment file. ClickOnce is a per-user installation format that
    is often incompatible with the per-machine nature of virtual packages. A ClickOnce application also
    may try to automatically update itself, which can result in invalid versioning in virtual
    application management systems.
```

[1] is the name of a file in the project.

Description

ISVICE03 checks for the presence of ClickOnce deployment files. If your project includes ClickOnce deployment files, ISVICE03 occurs.

Corrective Action

Evaluate how important the ClickOnce deployment files are to the application so that you can determine if it matters whether the ClickOnce installation behaves as intended. If the ClickOnce deployment files are unimportant, it is probably safe to deploy a virtualized version of the product with slightly reduced functionality. If the ClickOnce deployment files are important, a virtualized version of the product may not function well.

ISVICE04



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE04 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- ThinApp
- XenApp

Message (Error)

```
The file [1] appears to be a part of an ASP.NET/IIS application. This is not supported by desktop
    application virtualization.
```

[1] is the name of a file in the project.

Description

ISVICE04 checks for the presence of files that are part of ASP.NET or IIS applications.

Corrective Action

If the ASP.NET or IIS application is not an important part of the product, or if it can be separately installed locally on target systems, you can ignore this validation error.

ISVICE05



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE05 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- ThinApp
- XenApp

Message (Error)

```
The file [1] appears to be a WMI provider. This is not supported by desktop application virtualization.
    If the provider is not a critical piece of the package, it may be possible to use the rest of the
    package as a virtual application.
```

[1] is the name of a file in the project.

Description

ISVICE05 checks for the presence of files that are part of Windows Management Instrumentation (WMI) providers.

Corrective Action

If the WMI provider is not an important part of the product, or if it can be separately installed locally on target systems, you can ignore this validation error.

ISVICE06



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE06 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- Server App-V
- ThinApp
- XenApp

Message (Error)

```
The file [1] appears to be part of a J2EE application server. This is not supported by desktop
    application virtualization.
```

[1] is the name of a file in the project.

Description

ISVICE06 checks for the presence of files that are part of J2EE application servers.

Corrective Action

If the J2EE application is not an important part of the product, or if it can be separately installed locally on target systems, you can ignore this validation error.

ISVICE07



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *MSI Database*

ISVICE07 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- Server App-V
- ThinApp
- XenApp

Message (Warning)

```
The file [1] appears to be a part of an application that is known to not be a good candidate for
    virtualization; however, it may be possible to virtualize the rest of the package and install the
    unsuitable piece physically.
```

[1] is the name of a file in the project.

Description

ISVICE07 checks for the presence of files that are part of applications that may not work when virtualized. This includes files that are from applications such as SQL Server.

Corrective Action

If the unsupported application is not an important part of the product, or if it can be separately installed locally on target systems, you can ignore this validation error.

ISVICE08



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE08 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- Server App-V
- ThinApp
- XenApp

Message (Error)

```
The file [1] appears to be a part of an application that is known to not be a good candidate for
    virtualization.
```

[1] is the name of a file in the project.

Description

ISVICE08 checks for the presence of files that are part of applications that may not work when virtualized. This includes files that are from applications such as antivirus software or server software such as Exchange Server.

Corrective Action

If the unsupported application is not an important part of the product, or if it can be separately installed locally on target systems, you can ignore this validation error.

ISVICE09



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE09 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- ThinApp
- XenApp

Message (Error)

```
IIS [1] item [2] is included in the package. This is not supported by desktop application virtualization.
```

[1] is the name of a type of IIS item in the project, and [2] is the name of that item.

Description

ISVICE09 checks for the presence of InstallShield IIS support indicating that an ASP.NET or IIS application is installed.

Corrective Action

If the unsupported ASP.NET or IIS application is not an important part of the product, or if it can be separately installed locally on target systems, you can ignore this validation error.

ISVICE10



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *MSI Database*

ISVICE10 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- Server App-V
- ThinApp
- XenApp

Message (Error)

```
Component [1] contains a device driver installed through the MsiDriverPackages (DIFxApp) table. System-
    level drivers such as print drivers or USB device drivers do not work from a virtualized
    environment. It may be possible to extract this driver such that it can be installed locally on the
    machine and allow the rest of the package to be virtualized.
```

[1] is the name of a device driver that is included in the project.

Description

ISVICE10 checks for the presence of drivers that are configured to be installed through the **MsiDriverPackages** table.

Corrective Action

If your end users may be interested in a virtualized version of your product, and if your InstallShield project contains a driver, consider removing the driver from your project, and creating a separate installation for the driver. End users can then install the driver locally to their machines before using a virtualized version of your product.

ISVICE11



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE11 is available for the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- ThinApp
- XenApp

Message (Warning for App-V 4.x and App-V 5.x; Error for ThinApp and XenApp)

```
This package contains no shortcuts. Shortcuts are usually necessary to define the entry point into the
    virtual application.
```

Description

ISVICE11 checks the package to ensure that it contains at least one shortcut. Shortcuts provide the most visible entry points for launching applications in a virtual package. Most virtual packages should have at least one shortcut.

Corrective Action

If the package is intended to be used as a dependency by a different virtual package, you can ignore this ISVICE.

If the package is intended to be used as a plug-in, it may be necessary to create a shortcut to the application for which this is a plug-in.

If end users need to be able to launch the virtual application independently, consider adding a shortcut for the main executable file.

ISVICE12



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE12 is available for the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- Server App-V
- ThinApp
- XenApp

Message (Error for App-V 4.x, App-V 5.x, ThinApp, and XenApp; Warning for Server App-V)

```
This package contains a shell extension (key [1]). Shell extensions extend Windows Explorer and cannot
    be loaded from a virtual package. This extension may be critical to the use of this application, and
    if so this application will not function when virtualized. However, if this extension is non-
    critical, the application can still be used.
```

[1] is the key of a shell extension that is included in the project.

Description

ISVICE12 checks for the presence of shell extensions.

Corrective Action

Evaluate how important the shell extension is to the product so that you can determine if it matters whether the shell extension behaves as intended. If the shell extension is unimportant, it is probably safe to deploy a virtualized version of the product with slightly reduced functionality. If the shell extension is important, a virtualized version of the product may not function well.

ISVICE13



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE13 is available for the following virtualization technologies:

- App-V 4.x
- Server App-V
- ThinApp
- XenApp

Message (Warning)

```
This package registers a URL protocol (friendly name [1]).
```

[1] is the friendly name of a URL protocol.

Description

ISVICE13 checks for the presence of URL protocol handlers.

Corrective Action

Evaluate how important the registration of the URL protocol is to the product so that you can determine if it matters whether the product behaves as intended. If the registration of the URL protocol is unimportant, it is probably safe to deploy a virtualized version of the product with slightly reduced functionality. If the registration of the URL protocol is important, a virtualized version of the product may not function well.

Consider performing validation to see if App-V 5.x is a suitable virtualization technology for your product, since it has support for URL protocol handlers.

ISVICE14



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

- *MSI Database*

ISVICE14 is available for the following virtualization technologies:

- App-V 4.x
- Server App-V
- ThinApp
- XenApp

Message (Warning)

`This package registers its capabilities in the Default Programs list (key '%1').`

[1] is the name of the key that corresponds with registration information for an item in the Default Programs list.

Description

ISVICE14 checks for the presence of support for the Default Programs list.

Corrective Action

If the Default Programs list registration is not an important part of the product, you can ignore this validation error.

Consider performing validation to see if App-V 5.x is a suitable virtualization technology for your product, since it has support for registering items in the Default Programs list.

ISVICE15



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE15 is available for the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- ThinApp
- XenApp

Message (Error)

```
This package contains a service (name [1]), which starts at boot time. Virtualized services are limited
    to the lifetime of the virtual application so services that must start at boot time do not make good
    candidates for virtualization. It may be possible to extract this service such that it can be
    installed locally on the machine and allow the rest of the package to be virtualized.
```

[1] is the name of a service in the project.

Description

ISVICE15 checks for the presence of boot start services.

Corrective Action

Evaluate how important the service is to the product so that you can determine if it matters whether the product behaves as intended. If the service is unimportant, it is probably safe to deploy a virtualized version of the product with slightly reduced functionality.

If the service is important but you can separate it from the rest of the project, remove it from the project, and create a separate installation for the service. End users can then install the service locally to their machines before using a virtualized version of your product.

If the service is important but you cannot separate it from the rest of the project, a virtualized version of the product may not function well.

ISVICE16



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE16 is available for the following virtualization technologies:

- App-V 4.x
- Server App-V
- XenApp

Message (Error)

```
This package contains more than 4 GB of files. Since the target virtualization technology has a 4 GB size
    limit, this application cannot be successfully virtualized.
```

Description

ISVICE16 checks to ensure that the total installed file size is less than 4 GB. App-V 4.x, Server App-V, and XenApp do not support packages that contain more than 4 GB of files.

Corrective Action

Consider performing validation to see if App-V 5.x or ThinApp are suitable virtualization technologies for your product, since they support sizes larger than 4 GB.

ISVICE17



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE17 is available for the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- ThinApp
- XenApp

Message (Error)

```
This package contains COM+ data (component [1]), which is not supported, so the application may not work
    correctly if virtualized.
```

[1] is the name of a component in the project; the component contains COM+ data.

Description

ISVICE17 checks for use of the Windows Installer COM+ tables.

Corrective Action

Evaluate how important the COM+ application is to the product so that you can determine if it matters whether the product behaves as intended. If the COM+ application is unimportant, it is probably safe to deploy a virtualized version of the product with slightly reduced functionality. If the COM+ application is important, a virtualized version of the product may not function well.

ISVICE18



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE18 applies to the following virtualization technologies:

- App-V 4.x
- App-V 5.x
- ThinApp
- XenApp

Message (Error)

```
This package contains a COM DLL surrogate (registry key [1]), which is not supported, so the application
    may not work correctly if virtualized.
```

[1] is the name of a registry key in the project.

Description

ISVICE18 checks for the presence of COM DLL surrogate files.

Corrective Action

Evaluate how important the COM DLL surrogate is to the product so that you can determine if it matters whether the product behaves as intended. If the DLL surrogate is unimportant, it is probably safe to deploy a virtualized version of the product with slightly reduced functionality. If the DLL surrogate is important, a virtualized version of the product may not function well.

ISVICE19



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

ISVICE19 applies to the following virtualization technologies:

- ThinApp
- XenApp

Message (Error)

```
The package is 64-bit. ThinApp and XenApp do not support 64-bit applications.
```

Description

ISVICE19 checks to see if you are building a 64-bit package. ThinApp and XenApp do not support 64-bit applications.

Corrective Action

Consider performing validation to see if App-V or Server App-V is a suitable virtualization technology for your product, since it has 64-bit support.

InstallShield UWP App Suitability Suite



Project • *This information applies to the Basic MSI project type.*

InstallShield includes a set of validators called the InstallShield UWP App Suitability Suite. The InstallShield UWP App Suitability validators (ISUWPs) in this suite scan an .msi package for signs of items that are unsuitable for the UWP app package (.appx) format. To access this suite, on the **Build** menu, point to **Validation**, and then click **InstallShield UWP App Suitability Suite**. The suite provides a report in the Releases view that indicates all tests that found issues and for each issue, an associated column in the report indicates applicability to the known UWP app variants (Universal App, Desktop Bridge, Windows Store, WSA, and Nano Server). For traditional CUBs, these columns are not populated. The following ISUWP validations are available in InstallShield:

Table 4-20 • ISUWPs

ISUWP	Project Type	Technology	Description
ISUWP01	Basic MSI	UWP app	Checks for the presence of Windows services.
ISUWP02	Basic MSI	UWP app	Checks for the presence of drivers installed with the MsiDriverPackages table.
ISUWP03	Basic MSI	UWP app	Checks for the presence of shortcuts. InstallShield requires shortcuts to identify applications. Without shortcuts to identify applications, the package is invalid as there are no ways to start your app.
ISUWP04	Basic MSI	UWP app	Checks for the presence of multiple shortcuts for Windows Store applications. InstallShield turns shortcuts into applications, and the Windows Store may disallow multiple applications.
ISUWP05	Basic MSI	UWP app	Checks for the presence of shortcuts for Windows Nano Server applications. InstallShield will turn any shortcut into a tile if it points to an .exe installed by the package; however Windows Nano Server cannot display application tiles.
ISUWP06	Basic MSI	UWP app	Checks for the presence of 16-bit files. All server installations are 64-bit platforms and 64-bit platforms do not support 16-bit files..
ISUWP07	Basic MSI	UWP app	Checks for the presence of 32-bit files. All server installations are 64-bit platforms and 64-bit platforms do not support 32-bit files.
ISUWP08	Basic MSI	UWP app	Checks for the presence of applications that require elevated privileges.
ISUWP09	Basic MSI	UWP app	Checks for the presence of applications that are allowed to bypass UI protection levels.
ISUWP10	Basic MSI	UWP app	Checks for the presence of COM registration.
ISUWP11	Basic MSI	UWP app	Checks for the presence of executable files that set a custom Application User Model ID.

Table 4-20 • ISUWPs

ISUWP	Project Type	Technology	Description
ISUWP12	Basic MSI	UWP app	Checks for the presence of .NET assemblies that are incompatible with Windows Nano Server.
ISUWP13	Basic MSI	UWP app	Checks for the presence of .NET assemblies that do not support the latest version of .NET Framework.
ISUWP14	Basic MSI	UWP app	Checks for the presence of files that are part of ASP.NET/IIS applications.
ISUWP15	Basic MSI	UWP app	Checks for the presence of InstallShield IIS support indicating that an ASP.NET/IIS application is installed.
ISUWP16	Basic MSI	UWP app	Checks for the presence of built-in InstallShield support that executes SQL scripts.
ISUWP17	Basic MSI	UWP app	Checks for the presence of built-in InstallShield support that modifies text files.
ISUWP18	Basic MSI	UWP app	Checks for the presence of built-in InstallShield support that modifies XML files.
ISUWP19	Basic MSI	UWP app	Checks for the presence of built-in InstallShield support that creates scheduled tasks.
ISUWP20	Basic MSI	UWP app	Checks for the presence of file type associations that are reserved or forbidden.

ISUWP01

ISUWP01 applies to UWP app scenarios.

Message (Error)

```
This package contains one or more Windows services. UWP app packages that target other devices than
    Windows Server do not support Windows services.
```

Description

ISUWP01 checks for the presence of Windows services.

Corrective Action

If one or more Windows services are required by the application, then server extensions and WSA are required. Including desktop extensions is not supported in this scenario.

ISUWP02

ISUWP02 applies to UWP app scenarios.

Message (Error)

```
Component [1] contains a device driver installed through the MsiDriverPackages (DIFxApp) table. UWP app
    packages do not support device drivers.
```

[1] is the name of a component in the project.

Description

ISUWP02 checks for the presence of drivers installed with the MsiDriverPackages table.

Corrective Action

If the driver is required, this application cannot be supported purely by UWP apps.

ISUWP03

ISUWP03 applies to UWP app scenarios.

Message (Warning)

```
This package contains no shortcuts. Shortcuts are usually necessary to define the entry point into the
    application.
```

Description

ISUWP03 checks for the presence of shortcuts. InstallShield requires shortcuts to identify applications. Without shortcuts to identify applications, the package is invalid as there are no ways to start your app.

ISUWP04

ISUWP04 applies to UWP app scenarios.

Message (Warning)

```
This package contains more than one shortcut. Windows Store may not allow more than one application.
```

Description

ISUWP04 checks for the presence of multiple shortcuts for Windows Store applications. InstallShield turns shortcuts into applications, and the Windows Store may disallow multiple applications.

Corrective Action

If you wish to distribute this application on the Windows Store, you may need to split the application into multiple UWP app packages.

ISUWP05

ISUWP05 applies to UWP app scenarios.

Message (Warning)

```
This package contains one or more shortcuts. Windows Nano Server cannot show application tiles.
```

Description

ISUWP05 checks for the presence of shortcuts for Windows Nano Server applications. InstallShield will turn any shortcut into a tile if it points to an .exe installed by the package; however Windows Nano Server cannot display application tiles.

Corrective Action

This warning alerts you to a known item that may not function as expected.

ISUWP06

ISUWP06 applies to UWP app scenarios.

Message (Warning)

```
File [1] in Component [2] is a 16-bit file. 64-bit systems do not support 16-bit applications.
```

[1] is the name of a file in the project.

[2] is the name of a component in the project.

Description

ISUWP06 checks for the presence of 16-bit files. All server installations are 64-bit platforms and 64-bit platforms do not support 16-bit files.

Corrective Action

A corrective action is only necessary if the file is loaded and executed at run-time. Data files that happen to be 16-bit or 32-bit are not a problem. If the flagged 16-bit file is loaded and executed at run-time, replace it with a 64-bit file.

ISUWP07

ISUWP07 applies to UWP app scenarios.

Message (Warning)

```
File [1] in Component [2] is a 32-bit file. 64-bit systems may not support 32-bit applications.
```

[1] is the name of a file in the project.

[2] is the name of a component in the project.

Description

ISUWP07 checks for the presence of 32-bit files. All server installations are 64-bit platforms and 64-bit platforms do not support 32-bit files.

Corrective Action

A corrective action is only necessary if the file is loaded and executed at run-time. Data files that happen to be 16-bit or 32-bit are not a problem. If the flagged 32-bit file is loaded and executed at run-time, replace it with a 64-bit file.

ISUWP08

ISUWP08 applies to UWP app scenarios.

Message (Error)

```
File [1] in Component [2] is an application executable file that requires elevated privileges. Windows
    Store does not allow applications that require administrative privileges.
```

[1] is the name of a file in the project.

[2] is the name of a component in the project.

Description

ISUWP08 checks for the presence of applications that require elevated privileges.

Corrective Action

Remove the requirement for elevated privileges. An app may be using elevated privileges to subvert best-practices about handling files or registry keys, and such subversion will not work in a UWP app.

ISUWP09

ISUWP09 applies to UWP app scenarios.

Message (Error)

```
File [1] in Component [2] is an application executable file that is allowed to bypass UI protection
    levels to drive input to higher privilege windows on the desktop. UWP app packages do not support
    applications that are allowed to bypass UI protection levels.
```

[1] is the name of a file in the project.

[2] is the name of a component in the project.

Description

ISUWP09 checks for the presence of applications that are allowed to bypass UI protection levels.

Corrective Action

Very few applications require bypassing UI protection levels, but those that do are not currently supported by UWP apps. If an application requires bypassing UI protection levels, consider using a different package format than a UWP app package.

ISUWP10

ISUWP10 applies to UWP app scenarios.

Messages (Warnings)

Message 1 (Warning)

```
Table [1] contains one or more entries. Applications in UWP app packages cannot expose COM servers to
    other applications.
```

[1] is a table in your project.

Message 2 (Warning)

```
Registry key [1] in Component [2] is COM data. Applications in UWP app packages cannot expose COM servers
    to other applications.
```

[1] is the name of a registry key in the project.

[2] is the name of a component in the project.

Description

ISUWP10 checks for the presence of COM registration.

Corrective Action

COM registration may or may not be a problem. Internal use of COM is fully supported by the Desktop Bridge, but no extension currently supports public registration (for use by other apps). If the latter is required, a UWP app package is unsuitable.

ISUWP11

ISUWP11 applies to UWP app scenarios.

Message (Error)

```
File [1] in Component [2] imports API [3]. Applications in UWP app packages are not allowed to set a
    custom Application User Model ID.
```

[1] is the name of a file in the project.

[2] is the name of a component in the project.

[3] is the name of an API.

Description

ISUWP11 checks for the presence of executable files that set a custom Application User Model ID.

Corrective Action

Code-based removal of the offending API call is required. A UWP app will set an Application User Model ID for you.

ISUWP12

ISUWP12 applies to UWP app scenarios.

Messages

Message 1 (Error)

```
File [1] in Component [2] is a .NET assembly that was compiled against .NET Framework. Windows Nano
    Server supports only .NET Core applications
```

[1] is the name of a file in the project.

[2] is the name of a component in the project.

Message 2 (Error)

```
File [1] in Component [2] is a .NET assembly that was compiled against [3]. Windows Nano Server supports
    only .NET Core applications.
```

[1] is the name of a file in the project.

[2] is the name of a component in the project.

[3] is the name of a framework in the project.

Description

ISUWP12 checks for the presence of .NET assemblies that are incompatible with Windows Nano Server.

Corrective Action

Rebuild your app against .NET Core if you want to support Nano Server.

ISUWP13

ISUWP13 applies to UWP app scenarios.

Message (Warning)

```
File [1] in Component [2] is a .NET assembly that does not support the latest version of .NET Framework.
    Windows Store applications always run on the latest version of .NET Framework.
```

[1] is the name of a file in the project.

[2] is the name of a component in the project.

Description

ISUWP13 checks for the presence of .NET assemblies that do not support the latest version of .NET Framework.

Corrective Action

Rebuild or adjust your **app.config**.

ISUWP14

ISUWP14 applies to UWP app scenarios.

Message (Error)

```
The file [1] appears to be a part of an ASP.NET/IIS application. This is not supported by applications
    in UWP app package.
```

[1] is the name of a file in the project.

Description

ISUWP14 checks for the presence of files that are part of ASP.NET/IIS applications.

Corrective Action

Because files that are part of ASP.NET/IIS applications are not supported by UWP apps, you will need to remove the file from the package or use an alternative package format.

ISUWP15

ISUWP15 applies to UWP app scenarios.

Message (Error)

```
IIS [1] item [2] is included in the package. This is not supported by applications in UWP app packages.
```

[1] is the name of a type of IIS item in the project, and [2] is the name of that item.

Description

ISUWP15 for the presence of InstallShield IIS support indicating that an ASP.NET/IIS application is installed.

Corrective Action

Because files that are part of ASP.NET/IIS applications are not supported by UWP apps, you need to remove the file from the package or use an alternative package format.

ISUWP16

ISUWP16 applies to UWP app scenarios.

Message (Error)

```
This package contains one or more SQL scripts that run during an installation. UWP app packages do not
    support executing SQL scripts.
```

Description

ISUWP16 checks for the presence of built-in InstallShield support that executes SQL scripts.

Corrective Action

The flagged configuration must be moved from the installation to the application in order to support the use of a UWP app package.

ISUWP17

ISUWP17 applies to UWP app scenarios.

Message (Error)

```
This package contains one or more InstallShield Text File Changes items that are performed during an
    installation. UWP app packages do not support modifying text files.
```

Description

ISUWP17 for the presence of built-in InstallShield support that modifies text files.

Corrective Action

The flagged configuration must be moved from the installation to the application in order to support the use of a UWP app package.

ISUWP18

ISUWP18 applies to UWP app scenarios.

Message (Error)

```
This package contains one or more InstallShield XML File Changes items that are performed during an
    installation. UWP app packages do not support modifying XML files.
```

Description

ISUWP18 checks for the presence of built-in InstallShield support that modifies XML files.

Corrective Action

The flagged configuration must be moved from the installation to the application in order to support the use of a UWP app package.

ISUWP19

ISUWP19 applies to UWP app scenarios.

Message (Error)

```
This package contains one or more InstallShield Scheduled Tasks items that are performed during an
    installation. UWP app packages do not support creating scheduled tasks.
```

Description

ISUWP19 checks for the presence of built-in InstallShield support that creates scheduled tasks.

Corrective Action

The flagged configuration must be moved from the installation to the application in order to support the use of a UWP app package.

ISUWP20

ISUWP20 applies to UWP app scenarios.

Message (Error)

```
Extension [1] in Component [2] is a file type that is reserved or forbidden for UWP app packages.
```

[1] is the name of an extension in the project and [2] is the name of the component it is in.

Description

ISUWP20 checks for the presence of file type associations that are reserved or forbidden.

Corrective Action

Remove the flagged reserved or forbidden file type associations in order to support the use of a UWP app package.

InstallShield Best Practice Suite



Edition • *The InstallShield Best Practice Suite is available in the Premier edition of InstallShield.*



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Note that some of the ISBPs apply to only Basic MSI and MSI Database projects.

InstallShield includes a set of validators called the InstallShield Best Practice Suite. The InstallShield Best Practice (ISBP) validators in this suite alert you if your installation violates best-practice guidelines.

The following table lists the ISBPs that are available in InstallShield.

Table 4-21 • ISBPs

ISICE	Project	Description
ISBP01	Basic MSI, InstallScript MSI, MSI Database	Verifies that no feature is named “ALL”.
ISBP02	Basic MSI, InstallScript MSI, MSI Database	Verifies that no directory is named “DATABASE”.
ISBP03	Basic MSI, MSI Database	Verifies that no ComboBox is shorter than 50 units.
ISBP04	Basic MSI, MSI Database	Verifies that properties used on dialogs are secure or restricted public properties.
ISBP05	Basic MSI, MSI Database	Verifies that no ControlEvent condition is NULL.
ISBP06	Basic MSI, InstallScript MSI, MSI Database	Verifies that InstallUISequence custom actions are also sequenced in the InstallExecuteSequence.
ISBP07	Basic MSI, InstallScript MSI, MSI Database	Verifies that all features have associated components and all components are associated with features.
ISBP08	Basic MSI, InstallScript MSI, MSI Database	Verifies that ARPINSTALLLOCATION is set after CostFinalize in the InstallExecuteSequence.
ISBP09	Basic MSI, InstallScript MSI, MSI Database	Verifies that LIMITUI is not set without ARPNOMODIFY.
ISBP10	Basic MSI, InstallScript MSI, MSI Database	Verifies that AppSearch properties are secure or restricted public properties.
ISBP11	Basic MSI, InstallScript MSI, MSI Database	Verifies that no precompiled .NET assemblies are being distributed.
ISBP12	Basic MSI, InstallScript MSI, MSI Database	Verifies that no file is self-registered.

Table 4-21 • ISBPs (cont.)

ISICE	Project	Description
ISBP13	Basic MSI, MSI Database	Verifies that properties set by dialog controls and used in the installation have a default value.
ISBP14	Basic MSI, InstallScript MSI, MSI Database	Verifies that each file has the correct version information or an MsiFileHash entry.
ISBP15	Basic MSI, MSI Database	Verifies that no RadioButtonGroup has Text defined.
ISBP16	Basic MSI, InstallScript MSI, MSI Database	Verifies that each component with a 64-bit destination is marked as a 64-bit component.
ISBP17	Basic MSI, InstallScript MSI, MSI Database	Verifies that no .hlp files or WinHelp run-time files are included in the installation.
ISBP18	Basic MSI, InstallScript MSI, MSI Database	Verifies that no obsolete APIs are imported.
ISBP19	Basic MSI, InstallScript MSI, MSI Database	Verifies that no deprecated APIs are imported.
ISBP20	Basic MSI, InstallScript MSI, MSI Database	Verifies that no registry entries contained in the Registry table attempt to remove root-level registry keys (such as HKLM\Software) or other keys that would cause adverse issues on target machines.

ISBP01



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
Feature ALL conflicts with the installation meta-feature ALL.
```

Description

ISBP01 verifies that your installation does not have a feature called **ALL** in uppercase, lowercase, or mixed case letters. In the context of features, Windows Installer reserves the word **ALL** for use as a valid value for feature-state properties such as **ADVERTISE**, **REINSTALL**, and **ADDLOCAL**, which can contain the names of specific features, as well as the word **ALL** to indicate all features. Therefore, none of the individual features in a project should be called **ALL**.

Corrective Action

To resolve this error, change the name of the feature to something other than ALL. For more information, see Creating Features.

You can also resolve this error by clicking the ISBP01 error message in the Output window. InstallShield displays the **Feature** table in Direct Editor view, and highlights the row that contains the feature named **ALL**. To rename the feature, select the current name and then type a new one.

ISBP02



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
Directory DATABASE conflicts with Windows Installer's undocumented directory DATABASE.
```

Description

ISBP02 verifies that your installation does not have a directory called **DATABASE** in uppercase, lowercase, or mixed case letters. Windows Installer reserves this directory name.

Corrective Action

To resolve this error, click the ISBP02 error message in the Output window. InstallShield displays the **Directory** table in Direct Editor view, and highlights the row that contains the directory named **DATABASE**. To rename the directory, select the current name and then type a new one.

ISBP03



Project *• This information applies to the following project types:*

- *Basic MSI*
- *MSI Database*

Message (Warning)

```
ComboBox [1] on Dialog [2] has a height less than 50. This results in a ComboBox too short to view
    multiple items at once.
```

[1] is the name of a combo box control, and [2] is the name of the dialog that has that control.

Description

If a combo box control has a height of less than 50 installer units, end users may be able to see only one item in the box at a time. To improve the usability, consider changing the height to 50 or higher.

Corrective Action

To resolve this warning, click the ISBP03 warning message in the Output window. InstallShield displays the **Control** table in Direct Editor view, and highlights the row that contains the combo box control. Then change the value in the Height column for that control to 50 or higher.

ISBP04



Project • *This information applies to the following project types:*

- *Basic MSI*
- *MSI Database*

Messages

Message 1 (Warning)

```
Dialog [1] Control [2] uses private property [3]. Its value will be unavailable to the execute sequence.
```

[1] is the name of a dialog in your project, [2] is the name of a control on that dialog, and [3] is the name of a private property that is being set or used by the control.

Message 2 (Warning)

```
Dialog [1] Control [2] uses insecure custom property [3]. Its value may be unavailable to the execute
    sequence.
```

[1] is the name of a dialog in your project, [2] is the name of a control on that dialog, and [3] is the name of a property that is used by that control.

Description

You cannot set a private property in the user interface sequence of the installation and then pass the value of that private property to the execute sequence. Therefore, ISBP04 verifies that your installation does not have any dialog controls that use private properties. If a dialog control does use a private property, warning 1 occurs to alert you that you may need to take steps to allow a property's value to be passed to the execute sequence.

In addition, if you set a public property in the user interface sequence of an installation that requests elevated privileges for the execute sequence, and you want to pass the property's value to the execute sequence, the property must be listed as a value for the **SecureCustomProperties** property, or it must be a restricted public property. Therefore, ISBP04 also verifies that if your installation has dialog controls that use custom public

properties, the custom public properties are listed as values for the **SecureCustomProperties** property. If the custom public properties are not listed in the **SecureCustomProperties** property, warning 2 occurs to alert you that you may need to take steps to allow a property's value to be passed to the execute sequence.

Corrective Action

If you do not need to use the property in the execute sequence, you can disregard the warning message. However, if you do need to use the property in this sequence, you need to resolve the warning.

If you need to resolve warning 1, change the private property to a public property. To do so, find the dialog control in the Dialogs view; for the Property property of that control, use all uppercase letters, since public properties must be in all uppercase. Also update all other instances of that property name in your project so that it matches.

If you need to resolve warning 2, open the Property Manager view. For the Value column of the **SecureCustomProperties** property, add the name of the property that is mentioned in the warning message. If this property contains more than one property, separate each one with a semicolon (;).

ISBP05



Project • *This information applies to the following project types:*

- *Basic MSI*
- *MSI Database*

Message (Warning)

```
Dialog [1] Control [2] has a NULL condition. To ensure the control event always runs, use a condition of
    1.
```

[1] is the name of a dialog in your project, and [2] is the name of a control on that dialog.

Description

ISBP05 verifies that each event's condition for each dialog control is not null. This helps to ensure that Windows Installer triggers the control's events under the proper circumstances.

Corrective Action

If Windows Installer should trigger the event that has a null condition only if no other event for that control is triggered, you can disregard the warning message.

If you do need to resolve this warning, locate the dialog in the Dialogs view. Click the Behavior item under the dialog. In the list of controls, click the one that is mentioned in the warning message. Ensure that in the lower-right pane, the Events tab is displayed. Then, in the grid in the upper-right pane, enter a condition for the event. To ensure that Windows Installer triggers the event, enter 1 as the condition.

ISBP06



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Warning)

```
Custom Action [1] is scheduled in the InstallUISequence but not the InstallExecuteSequence. It will not
    run during a silent or reduced UI installation.
```

[1] is the name of a custom action in your project.

Description

Actions that are scheduled for the Execute sequence during the Installation sequence do not run during silent, basic UI, or reduced UI installations.

Corrective Action

If Windows Installer should launch the custom action during silent, basic UI, and reduced UI installations, consider scheduling the custom action for the installation execute sequence—in addition to or instead of the installation UI sequence.

To reschedule the action, open the Custom Actions and Sequences view, and find the action mentioned in the warning message. Then drag it from the Installation sequence's User Interface sequence to the Installation sequence's Execute sequence.

To schedule the action in both sequences, copy the action in the Installation sequence's User Interface sequence to the Installation sequence's Execute sequence. You can do this in the Custom Actions and Sequences view by pressing and holding CTRL while dragging the action from the former sequence to the latter sequence.

If Windows Installer should not launch the custom action during silent, basic UI, and reduced UI installations, you can ignore this warning.

ISBP07



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Messages

Message 1 (Warning)

```
Feature [1] has no components.
```

[1] is the name of a feature in your project.

Message 2 (Warning)

```
Component [1] is not associated with any features.
```

[1] is the name of a component in your project.

Description

ISBP07 verifies that every component in your project belongs to at least one feature. If you do not associate a component with at least one feature, Windows Installer cannot install the component.

ISBP07 also verifies that every feature in your project contains at least one component. Components contain the installation elements, such as files, shortcuts, and registry entries; if a feature does not contain a component, it does not contain anything to install.

Corrective Action

To resolve either of these warnings, use the Setup Design view to associate components with features. To learn more, see Associating Existing Components with Features.

ISBP08



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Warning)

```
There does not appear to be a Type 51 action setting ARPINSTALLLOCATION after CostFinalize in the
    InstallExecuteSequence.
```

Description

The **ARPINSTALLLOCATION** property specifies the fully qualified path to the primary destination folder of a product. Windows Installer writes this value to the Uninstall registry key.

The **ARPINSTALLLOCATION** property is typically set by a set-a-property type of custom action (type 51).

The SetARPINSTALLLOCATION custom action is a built-in InstallShield custom action that is added automatically to Basic MSI and InstallScript MSI projects. If you delete this custom action from your project, you may encounter the ISBP08 warning.

Corrective Action

To resolve this warning, add to your project a custom action with the following settings:

- Property Name: ARPINSTALLLOCATION
- Property Value: [INSTALLDIR]
- Execution Scheduling: Always execute
- Install Exec Sequence: After CostFinalize



***Note** • As a best practice, any actions after CostFinalize should be sequenced after MigrateFeatureStates when feature migration is selected on a major upgrade item.*

- Install Exec Condition: Not Installed

Leave the default values for all of the other settings.

For more information, see Creating Custom Actions in the Custom Actions and Sequences View (or the Custom Actions View).

ISBP09



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
The property LIMITUI has been set, but ARPNOMODIFY has not. This can lead to undesirable behavior with
    Add or Remove Programs.
```

Description

If you set the **LIMITUI** property in an installation, you should also set the **ARPNOMODIFY** property.

The **LIMITUI** property sets the user interface level to basic. An installation run with a basic UI typically displays only progress messages to end users. It does not allow end users to select features or provide other feedback.

The **ARPNOMODIFY** property prevents end users from modifying the product through Add or Remove Programs.

Corrective Action

To resolve this error, add the **ARPNOMODIFY** property to your project through the Property Manager view, and set its value to 1. For more information, see Creating Properties in Windows Installer–Based Projects.

ISBP10



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Warning)

```
AppSearch [1] uses insecure custom property [2]. Its value may be unavailable to the execute sequence.
```

[1] is the name of a system search in your project, and [2] is the name of a property that is used by that system search.

Description

If you set a public property in the user interface sequence of an installation that requests elevated privileges for the execute sequence, and you want to pass the property's value to the execute sequence, the property must be listed as values for the **SecureCustomProperties** property, or it must be a restricted public property. Therefore, ISBP10 verifies that if the AppSearch table in your project contains custom public properties, the custom public properties are listed as values for the **SecureCustomProperties** property. If the custom public properties are not listed in the **SecureCustomProperties** property, the warning occurs to alert you that you may need to take steps to allow a property's value to be passed to the execute sequence.

Corrective Action

If you do not need to use the property in the execute sequence, you can disregard the warning message. However, if you do need to use the property in this sequence, you need to resolve the warning.

To resolve this warning, open the Property Manager view. For the Value column of the **SecureCustomProperties** property, add the name of the property that is mentioned in the warning message. If this property contains more than one property, separate each one with a semicolon (;).

ISBP11



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Warning)

```
File [1] appears to be a precompiled native image of a .NET assembly. This may decrease the efficiency
    compared to precompiling on the target machine.
```

[1] is the name of a precompiled native image (.ni) of a .NET assembly.

Description

ISBP11 verifies that your project does not include a precompiled native image of a .NET assembly.

Corrective Action

To resolve this warning, replace the native image file mentioned in the message with the appropriate .NET assembly file.

If you want the assembly to be compiled to machine code during the installation, select Yes for the .NET Precompile Assembly setting of the file's component.

Precompilation, or just-in-time compilation, takes into account the fact that some code might never be called during execution. It converts the assembly as it is needed during execution and stores the resulting native code so it is accessible for subsequent calls. Precompiling on the target machine allows the process to take advantage of the exact architecture of the machine on which it will be running.

ISBP12



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Messages

Message 1 (Error)

```
Custom Action [1] appears to invoke self-registration via RegSvr32. Best practices encourage authoring
    COM and Registry table data into the installer package.
```

[1] is the name of a custom action in your project that may be registering a file through **RegSvr32.exe** at installation time.

Message 2 (Error)

```
File [1] is self-registered. Best practices encourage authoring COM and Registry table data into the
    installer package.
```

[1] is the name of a file that is marked as self-registering.

Description

Although InstallShield lets you designate that a COM server is self-registering, the preferred method of registering a COM server is to extract the COM information from the file at build time or design time; with this method, InstallShield writes the COM class information to the **Class**, **ProgID**, and **Registry** tables of the .msi database. Self-registration has several limitations; for details, see Self-Registering COM Servers.

Corrective Action

To resolve error 1, remove the custom action that is referenced in the error message, and configure COM extraction for the file as appropriate.

To resolve error 2, configure COM extraction for the file as appropriate.

To learn more, see Extracting COM Information from a COM Server.

ISBP13



Project *• This information applies to the following project types:*

- *Basic MSI*
- *MSI Database*

Message (Error)

```
Property [1] has no default, is set by a dialog control, and is used in table [2]. This may not be
    populated in a silent or reduced UI installation.
```

[1] is the name of a property in your project, and [2] is the name of the table that uses that property.

Description

In most cases, each property in the **Property** table should have a default value. The default value is used if end users run the installation in silent, reduced UI, or basic UI mode.

Corrective Action

If you do not need to add a default value for the property, you can disregard the error message.

To resolve this error, open the Property Manager view. Enter a value in the Value column for the property that is specified in the error message.

ISBP14



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Messages

Message 1 (Error)

```
File [1] has neither a file version nor an MsiFileHash record. This may require the source package when
    installing a patch.
```

[1] is the name of file in your project.

Message 2 (Error)

```
File [1] is recorded as version [2] but is actually version [3]. This may require the source package when
    installing a patch.
```

[1] is the name of file in your project. [2] is the version number that is configured for the file, but [3] is the actual version number.

Description

ISBP14 verifies that each file in the project has the correct version information or an MsiFileHash entry. Using the correct version information in versioned files and MsiFileHash entries for unversioned files helps Windows Installer to detect and eliminate unnecessary file copying. It also helps ensure that the source package is not required to apply a patch to an earlier version of a product.

Corrective Action

To resolve error 1, consider adding the file version number to the file. If the file is an unversioned file, configure the file so that a file hash is used.

InstallShield enables you to override a file's version information. Error 2 may occur if the file's properties were overridden. To resolve this error, consider removing the version override.

To configure the settings for a file, open the Files and Folders view. In the **Destination computer's files** pane, locate the file, right-click it, and click Properties. The Properties dialog box opens, enabling you to configure version information and specifying whether file hashing should be used.

ISBP15



Project • *This information applies to the following project types:*

- *Basic MSI*
- *MSI Database*

Message (Error)

```
Control [2] on the Dialog [1] is a borderless RadioButtonGroup with overlapping controls and data in the
    Text column. This may cause repaint issues on some systems.
```

Description

ISBP15 verifies that if you have a borderless radio button group control on a dialog and one or more other controls overlap the radio button group, the radio button group does not have anything specified for its Text attribute. If it does, it may cause repaint issues for some versions of Windows Installer.

Corrective Action

To resolve this error, click the ISBP15 error message in the Output window. InstallShield displays the **Control** table in Direct Editor view, and highlights the row that contains the radio button control that is mentioned in the error message. Delete the string in the Text column for that radio button control.

As an alternative, you can change the radio button group control so that it uses a border. To do this, open the Dialogs view and find the dialog mentioned in the error message. Click the radio button control, and change its Has Border property to True.

ISBP16



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
Component [1] installed to 64-bit location [2] is not marked with the 64-bit component attribute. This
    may allow files to be installed to an incorrect location.
```

[1] is the name of a component, and [2] is the destination folder for the component's files.

Description

If a component is not configured to be 64 bit, Windows Installer may not install the component's files to the appropriate 64-bit location.

Corrective Action

To specify that a component is 64 bit, select Yes for the component's 64-Bit Component setting.

For more information about the 64-Bit Component setting, as well as additional component settings, see Component Settings.

ISBP17



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Messages

Message 1 (Error)

```
File [1] in Component [2] is a WinHelp file and may not be distributed to Windows Vista or later.
```

[1] is the name of an .hlp file in your project, and [2] is the name of the component that contains the .hlp file.

Message 2 (Error)

```
File [1] in Component [2] is a WinHelp run-time file and may not be distributed to Windows Vista or
    later.
```

[1] is the name of a Windows Help application file in your project, and [2] is the name of the component that contains that file.

Description

ISBP17 verifies that .hlp files are not included in your project because Windows Vista and later do not support this type of file.

In addition, ISBP17 verifies that **WinHlp32.exe** and **WinHelp.exe** are not included in the project because these applications should not be redistributed.

Corrective Action

To resolve this validation error, remove the help file from your project. Consider converting your help system to a different file format such as .chm, .htm, or .xml. Note that you would need to change your calls from the WinHelp API to the new help system.

ISBP18



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
File [1] in Component [2] imports obsolete API [3].
```

[1] is the name of a file in your project, and [2] is the name of the component that contains the file. [3] is the name of the obsolete API that the file uses.

Description

ISBP18 verifies that none of the .dll or .exe files in your project use obsolete kernel APIs.

Corrective Action

To resolve this validation error, replace the file with an updated version that does not use the obsolete API. Note that to create the updated version, you may need to rewrite and rebuild the .exe or .dll file. If the .exe or .dll file is from a third-party vendor, contact the vendor to request an updated version.

ISBP19



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Message (Error)

```
File [1] in Component [2] imports deprecated API [3].
```

[1] is the name of a file in your project, and [2] is the name of the component that contains the file. [3] is the name of the deprecated API that the file uses.

Description

ISBP19 verifies that none of the .dll or .exe files in your project use deprecated kernel APIs.

Corrective Action

To resolve this validation error, replace the file with an updated version that does not use the deprecated API. Note that to create the updated version, you may need to rewrite and rebuild the .exe or .dll file. If the .exe or .dll file is from a third-party vendor, contact the vendor to request an updated version.

ISBP20



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

Messages

Message 1 (Error)

```
Registry entry [1] contains an asterisk in the Name column. This will cause Windows Installer to attempt
    to remove a root registry key [2].
```

[1] is the name in the **Registry** table's Registry column for the registry entry row that has the asterisk (*). [2] is the name of the registry key that would be removed during uninstallation.

Message 2 (Warning)

```
Registry entry [1] installs to a key that is not recommended for installation with software
    applications. Installing this entry could have an adverse effect on target machines ([2]).
```

[1] is the name in the **Registry** table's Registry column for the registry entry row that may cause issues on the target system. [2] is the name of the registry key.

Description

ISBP20 verifies that your project does not include any registry entries that would remove root-level registry keys such as HKEY_LOCAL_MACHINE\SOFTWARE if your product is removed from a target machine. The following table shows an example of some registry data that would cause issues during uninstallation; in this example, the entire HKEY_LOCAL_MACHINE\SOFTWARE, and all of its subkeys and values, would be removed.

Table 4-22 • Example of Registry Data that Triggers an ISBP20 Error

Registry	Root	Key	Name	Value	Component
Registry2	2	SOFTWARE	*		MyComponent

ISBP20 also verifies that your project does not include other registry entries that would cause adverse issues on target machines.

Corrective Action

To resolve the validation error or warning, click the ISBP20 message in the Output window. InstallShield displays the **Registry** table in Direct Editor view, and highlights the row that contains the potentially problematic registry entry. To delete the registry entry, right-click the row and then click Drop Row(s).

Understanding When an Installation or Uninstallation Restarts the Target System



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI—if the InstallScript user interface (UI) style is the traditional style (which uses the InstallScript engine as an external UI handler)*

This information does not apply to InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

The BATCH_INSTALL system variable is set to a non-zero value to indicate that one or more operations need to be performed after the target system restarts. For example, BATCH_INSTALL can be set to a non-zero value if the installation determines that a file cannot be installed because the file already exists on the target system and it is locked.

When a file in an installation cannot be installed because it is locked, the installation automatically notifies the operating system that the file needs to be updated the next time that the system restarts. Because the file may need to be updated before additional Windows files are loaded, it is updated when Windows is loaded; it is not updated by the InstallScript installation.

BATCH_INSTALL can be set during an uninstallation in order to remove locked files after the system restarts. However, after the reboot, the cached installation files are no longer present. Therefore, the only reboot-related operations that can occur are those that the operating system can perform.

If BATCH_INSTALL is set to a non-zero value when the file transfer finishes, the following occurs:

- The installation does not attempt to self-register any files. This is because it may be necessary for all of the installation's files to be completely installed and updated before the files can be self-registered successfully. Note that this is true regardless of whether the files that could not be installed are self-registering, since the installation assumes that these files could be dependencies of other self-registering files. In this case, the installation records the files that need to be self-registered so that after the restart, the installation can self-register those files.
- The framework automatically calls **SdFinishReboot** (in the OnFirstUIAfter or OnMaintUIAfter events) to give the end user the option of automatically restarting the system after the installation completes.
- When the installation finishes, it creates an appropriate RunOnce registry value so that the installation runs after the restart; after the restart, the installation is run with the `/reboot` parameter.

  If the ALLUSERS script variable is set to 1, the RunOnce entry is created under HKEY_LOCAL_MACHINE; otherwise, the entry is created under HKEY_CURRENT_USER. Windows supports RunOnce entries under both of these locations.

  The RunOnce registry value's name is **InstallShield Setup %n**, where **%n** is the lowest integer available to make a unique key name. The registry value's data is the complete command line for the installation to run after restart; the command line includes the path and file name of **Setup.exe**.

  After restart, the installation is run from the installed Disk1 location—DISK1TARGET. Thus, for the installation to run after restart, the Disk1 files for the installation must be currently installed.

  Note that this does not apply to an uninstallation, since after the restart, the cached installation files are no longer present.

When the installation runs after the restart (that is, when it runs with the `/reboot` parameter), the following occurs:

- Any self-registering files that were noted by the installation as needing to be registered are registered.
- The **OnRebooted** event handler is called.

Note that for an uninstallation, the installation does not run after the restart; therefore none of these operations can occur.

Debugging Windows Installer–Based Installations

You can debug a Windows Installer–based release using the MSI Debugger. The MSI Debugger enables you to view and set Windows Installer properties as you step through the package's User Interface and Execute sequences.

The MSI Debugger runs through each action and dialog until it reaches your breakpoint, at which point it halts execution. Now, you can view and set properties in the Watch window and the Variable window. Finally, you can step through each of the remaining actions, or you can stop the debugger.



***Note** • Do not confuse the MSI Debugger with the InstallScript Debugger, since they have completely separate purposes. You cannot debug an installation package with the InstallScript Debugger, and you cannot debug an InstallScript custom action with the MSI Debugger.*

Starting the MSI Debugger

Before starting the MSI Debugger, you must have built a release and opened the MSI Debugger view in InstallShield. In addition, you should always set a breakpoint so that you can view the state of the installation at a particular action or dialog.

When you open the MSI Debugger and begin debugging, it acts on the release that currently has focus in the Releases view. If you have not yet built the release or if the release is compressed into **Setup.exe**, the debugger is blank and its toolbar and menu items are disabled. Switch to a built release or rebuild with the required options to debug the package.



Task ***To start debugging, do one of the following:***

- Click the **MSI Debugging** button on the MSI Debugger toolbar.
- Click the **Step Over** button on the MSI Debugger toolbar.
- On the **Build** menu, point to **MSI Debugger** and click **Start MSI Debugging**.
- On the **Build** menu, point to **MSI Debugger** and click **Step Over**.
- Press F5.
- Press F11.

The debugger runs the installation, returns focus to itself, and stops the execution of the installation when it reaches the breakpoint. Back in the debugger, you can do any of the following:

- View and set Windows Installer properties.
- Step through the rest of the installation.
- Exit the debugger and/or the installation.

Setting Breakpoints in the MSI Debugger



Task ***To set a breakpoint in the MSI Debugger:***

1. In the **Releases** view, select the release you want to debug.
2. Open the **MSI Debugger** view. The debugger displays two lists: first every standard and custom action in the User Interface sequence, and then every dialog in the installation.
3. Place the cursor on the line that contains the action or dialog on which you want the debugger to break.
4. Do one of the following to set the breakpoint on that line:
   - Click the **Toggle Breakpoint** button on the MSI Debugger toolbar.
   - On the **Build** menu, point to **MSI Debugger** and click **Toggle Breakpoint**.
   - Press F9.

You can set the breakpoint before you start debugging or while a debugging session is open.

When the debugger reaches the breakpoint, you can view and set properties or continue debugging.



***Tip** • Notice that the debugger lists the condition, if any, after each action. Set breakpoints carefully on conditional actions, because if the condition does not evaluate to true, the action is not executed and the debugger will not step in at that point.*

Removing Breakpoints



Task ***To remove a breakpoint:***

1. In the View List under **Additional Tools**, click **MSI Debugger**.
2. Place the cursor in a line with a breakpoint and click the **Toggle Breakpoint** button or press F9 to remove it.



Task ***To clear all breakpoints set in the debugger, do one of the following:***

- Click the **Remove All Breakpoints** button.
- Press SHIFT+F9.

Viewing and Setting Properties in the MSI Debugger

When the debugger stops at a breakpoint or at an action or dialog because you are stepping through the installation, you can view Windows Installer properties and change their values at run time in the Watch window or the Variable window.

The Variable Window

The Variable window displays every property in the database's **Property** table—exposed in InstallShield through Property Manager View—and its current value.

To change a property's value as the installation is running, edit the **Property Value** column.

If you do not see the Variable window, click Variable on the View menu.

The Watch Window

You can enter the name of any property in the Watch window to check its value at any point in the installation. For example, enter **TARGETDIR** in the name column to see what TARGETDIR resolves to at run time.

To change the property's value as the installation is running, edit the **Property Value** column.

If you do not see the Watch window, click Watch on the View menu.

Step Into Actions in the MSI Debugger

You can debug your code when you step into the custom action in the MSI Debugger. You can launch your registered debugger when you step into one of the following types of custom actions:

- Windows Installer .dll file custom action
- Standard .dll file custom action
- Visual Basic Script or JavaScript custom action
- InstallScript custom action



Task ***To step into the current action in the MSI Debugger, do any of the following:***

- Click the **Step Into** button on the MSI Debugger toolbar.
- On the **Build** menu, point to **MSI Debugger** and click **Step Into**.
- Press F11.

A dialog box opens and asks if you want to launch your registered debugger. Select Yes to launch the deubugger. Select No to step over to the next custom action.



Note *• Because InstallShield does not provide the source code for the entry point, you will see some machine code once the registered debugger is launched. You need to click the Step Into button twice to see your code.*

Step Through Actions in the MSI Debugger



Task ***To step over the current action or dialog and continue the execution in the MSI Debugger, do any of the following:***

- Click the **Step Over** button on the MSI Debugger toolbar.
- On the **Build** menu, point to **MSI Debugger** and click **Step Over**.
- Press F10.

Once the debugger reaches a breakpoint on a standard or custom action, it halts the execution of the installation and receives focus. You can then watch and set installer properties.

To continue stepping through each successive action, keep pressing F10. You may need to click Next on each dialog in the Setup wizard to progress through the sequence. To view the return value of an action, place the cursor over the action after it has executed. The return value is displayed as a tooltip. If the action executed successfully, the tool tip reads ERROR_SUCCESS.

Although setting a breakpoint on an action causes the debugger to break at each subsequent action, setting a breakpoint on a dialog takes you just to that dialog and then the debugger does not step in again until the next breakpoint.

Stopping the MSI Debugger



Task ***To discontinue debugging a release, do one of the following:***

- Click the **Stop Debugger** button on the MSI Debugger toolbar.
- On the **Build** menu, point to **MSI Debugger** and click **Stop MSI Debugging**.
- Press SHIFT+F5.

If the debugger is stopped at a break point on a dialog when you try to end the debugging session, InstallShield displays a message box with this message: "Please close all the dialogs to stop the debugging session." To close any open dialogs, first press F11 to step over the current breakpoint. Then, close any open dialogs.

Canceling out of the installation also stops the debugger.

Spanning Installations over Multiple Disks or CDs

As programs become larger, the need for disk spanning increases. Years ago, this meant shipping your product on multiple floppy disks. The standard is now to use CD-ROMs or DVDs. Although the storage space on a CD or on a DVD is significantly more than what is available on a floppy disk, many products require even more space. Multimedia tutorials, vast help libraries, and graphic-rich programs can result in a product that is larger than 650 MB—the size of a standard CD.

If your installation requires more than one CD, DVD, or custom-sized media disk, you need to span it across multiple disks.

Using the Release Wizard

The only way to define how your installation spans across multiple disks is to use the Release Wizard. The wizard's panels walk you through the process of creating your release for any size media that you choose, and it lets you specify how your files will span across multiple disks. You can also specify the compression to apply to these files.

The wizard offers you the choice of having InstallShield automatically span your installation across disks, if necessary, or you can customize how you want your installation files to be split. If you plan on customizing the way your installation spans across multiple disks, refer to Disk Spanning Rules to reduce problems with your installation.

Limitations

Due to limitations of the Windows Installer service, you cannot run multi-disk installations on a non-removable drive. For example, if your installation spans two CDs, you need to physically burn the CDs in order to test your installation. If you try to run it from a fixed drive, the installation will fail.

Compressing Multi-Disk Installations

Because neither **Setup.exe** nor your .msi file can be spanned across multiple disks, your source files need to be kept separate from these files. If you create a network installation, which has unlimited size, and specify that all the files should be compressed, these files are placed inside the .msi file or inside **Setup.exe** if you elected to create one. All other media types put **Setup.exe**, your .msi file, and the .cab files that contain all of your source files separately on the media.

For example, you might have an installation that contains three features—each containing a 1.5 MB file, **Setup.exe**, and the installation files for Windows NT—and you want to create a custom media type that is 2 MB in size. The build will span multiple disks. Disk one will contain **Setup.exe**, **InstMsiW.exe** (which contains the logic to install the Windows Installer service on Windows NT machines), **Setup.ini** (which is required for installations that include **Setup.exe**), and your .msi file. The remaining disks will contain .cab files that store compressed copies of all your source files.

Disk Spanning Rules

You can elect to have the Release Wizard automatically span your installation across as many disks as required. The wizard adheres to the following rules, which all multi-disk installations must follow. If you plan to customize how your installation spans across multiple disks, follow these rules:

- **Setup.exe must be on disk one.** This rule is applicable if you want to ship your installation on floppy disks and you need to ship **Setup.exe**. The size of **Setup.exe**—depending on whether you choose one for Windows 9x, Windows NT, or both—can be 1.31 MB to 2.58 MB in size. The installation file for the Windows Installer service is included in the build as well. This file—called **InstMsiW.exe** or **InstMsiA.exe**, depending on the platform you choose—must also reside on disk one.
- **The second file that must be included on disk one is the .msi file that you are building.** The total combined size of the .msi file, **Setup.exe**, and the Windows Installer installation files exceeds the maximum capacity of a standard 1.4 MB floppy disk. You have two options if you plan to distribute your setup on floppy disks:
  - You can distribute your installation without **Setup.exe**. This option is not available if you are creating an InstallScript MSI project.
  - The second and most feasible option is to use a distribution medium other than floppy disks.
- **Any transforms included in your installation must reside on the first disk.** For example, if you create an installation with multiple run-time languages, an .mst file is created for each of those languages. This file is applied to the installation database, and it provides the language selected in the Language Selection dialog.
- **Redistributables (including merge modules) included in your installation must be stored on the first disk.** Redistributables are streamed into the installation database while the script is built. This process cannot be completed if the redistributables are located on another disk.

Custom Disk Spanning

In the Release Wizard, you will come across a panel that asks you how you would like to handle disk spanning. You can choose to let the wizard automatically span your installation across disks if necessary, or you can manually define the disk spanning. Although you cannot indicate where every file is placed, you can define the disk on which each feature resides. Components and files are automatically placed on the same disk as their feature. There are other guidelines that InstallShield automatically follows, whether you choose to define the layout or let the wizard do it for you.

In addition to defining the features that reside on each disk, you can customize the volume label for each disk, create your own disk prompts, and choose to have the wizard enforce the size of your disks. The step-by-step instructions for customizing your disk layout are outlined in the Release Wizard documentation.

Disk Prompts

Disk prompts are displayed to end users when they need to place another disk in the drive to continue with the installation. Although a standard disk prompt is provided, you can provide a prompt that is specific to your product.

The string that is displayed actually comes from a number of sources throughout your project, as described below:

1. The complete prompt originates in the **Error** table (exposed in the Direct Editor) as error 1302.
2. This value comes from the project's list of string entries, and the default value for English is "Please insert the disk: *[2]*."
3. Windows Installer resolves *[2]* with the value of the property **DiskPrompt**, which is set to *[1]* in the Property Manager.
4. Finally, Windows Installer evaluates *[1]* to the string you enter into the Disk Prompt field.

In most cases, you will want to enter the same name as the disk volume. For example, assuming the defaults are unchanged, a Disk Prompt entry of **Disk8** would cause the user to be prompted with "Please enter the disk: Disk8."

You can use a question mark where the disk number will be displayed. This question mark automatically evaluates to the correct disk number. If you want to prompt your users with, "Please insert the disk: DISK4," you would enter **DISK?** for the fourth disk's name. You need to specify the disk prompt for each disk in your installation.

Volume Labels

The volume label is the name of the disk on which your installation resides. When the installation calls for a new disk, the Windows Installer engine verifies that the volume label of the disk matches the volume label that is stored in the installation database. If the labels do not match, the service assumes that the wrong disk has been inserted in the drive. You can change the volume label when you customize your disk spanning options. The default volume label is DISK1, DISK2, DISK3, and so on. When you build your installation, you will see these directories created in your output location. The disks on which you put your setup need to have the volume labels set to the same names as given by the wizard.



***Caution** • You cannot change a volume label after the installation has been built. If you do not specify the correct volume label, the Windows Installer service will not recognize the disk and the installation will fail.*

You can use a question mark to define the number of the disk. For example, if you want to set the volume label for your disks to **InstallShield Disk 1**, **InstallShield Disk 2**, and so on, you would rename every disk so that they read **InstallShield Disk ?**. At build time, this question mark evaluates to the disk number.

Enforce Disk Size

The Enforce Disk Size option helps you to ensure that your current layout fits on the disks that you have specified. If this option is not selected, and any disk that you want to create goes over the size of the media that you selected, the wizard automatically creates another disk and splits your features accordingly. If you select this option and the disk size exceeds the media size, the build will fail and you will receive build error 1531. You can then go back and trim down your feature size or try a different layout strategy.

Additional Information

If you add features to your installation after you have defined how you want your installation to span across multiple disks, those features are automatically added to the last disk of your installation. You can add new disks to your installation at any time by using the Release Wizard.

Creating a Setup Launcher



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Note that InstallScript installations always include a setup launcher.

Advanced UI and Suite/Advanced UI installations also always creates a setup launcher; they run .msi, .msp, InstallScript, and .exe packages. To learn more, see Creating Advanced UI and Suite/Advanced UI Installations.

InstallShield lets you specify whether you want your installation to include a **Setup.exe** setup launcher. A **Setup.exe** setup launcher is required in the following cases:

- You want to automatically update or install the Windows Installer engine on a target system, when necessary.
- You are building a multilanguage installation project and you want the language selection dialog to be displayed.
- Your project includes InstallShield prerequisites.
- Your project includes the .NET Framework.
- You are building a release for an InstallScript MSI project and it uses the traditional style for the user interface.
- You are building a release whose product configuration includes support for installing multiple instances of a product and you want the instance selection dialog to be displayed when appropriate.
- You are building a release for a Basic MSI project that includes billboards.

The **Setup.exe** setup launcher is a bootstrap application that manages the aforementioned scenarios.

The Setup.exe tab for a release in the Releases view is where you specify information such as whether you want to use a **Setup.exe** launcher. To learn more, see Setup.exe Tab for a Release.



Tip • *You can also specify setup launcher requirements in the Setup Launcher panel of the Release Wizard.*

Windows Installer and Setup.exe

If it is possible that Windows Installer is not present on a target system, or if your installation depends on certain functionality that is available in only a certain version of Windows Installer, InstallShield gives you the option of including with your installation a redistributable that installs Windows Installer. If you select this option, InstallShield creates a **Setup.exe** launcher that checks for the presence of Windows Installer on the target system. If Windows Installer is not installed, or a more recent version needs to be installed, **Setup.exe** launches the Windows Installer installation and then launches your installation package.

For more information, see Adding Windows Installer Redistributables to Projects.

Multilanguage Support and Setup.exe

Depending on your selections in the language-related settings on the Build tab for a release in the Releases view, **Setup.exe** also gives end users a choice of the language in which to run the installation.

If you plan to distribute your installation in more than one language, **Setup.exe** is required whenever you want to provide end users the option of selecting which language version of the installation they would like to run. To learn more about multilanguage support, see Creating Multilingual Installations.

InstallShield Prerequisites and Setup.exe

Projects that include InstallShield prerequisites require **Setup.exe** because **Setup.exe** checks to see if the target system meets the InstallShield prerequisite conditions. If the conditions are met, **Setup.exe** installs the InstallShield prerequisites. For more details, see Working with InstallShield Prerequisites that Are Included in Installation Projects.

.NET Framework and Setup.exe

Projects that include the .NET Framework require **Setup.exe** because **Setup.exe** checks the target system for the presence of the .NET Framework; if the appropriate version of the .NET Framework is not present, **Setup.exe** installs it.

To learn more about including the .NET Framework, see Adding .NET Framework Redistributables to Projects.

InstallScript MSI Projects and Setup.exe

The traditional-style InstallScript MSI installations require **Setup.exe** because **Setup.exe** initiates the InstallScript engine to display the user interface. For this traditional style, the InstallScript engine serves as an external UI handler.

Note that if you use the new style of UI for an InstallScript MSI installation, InstallShield embeds the InstallScript engine within the .msi package, and a setup launcher is not required.

For detailed information about these two styles, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

Multiple-Instance Support and Setup.exe

If you are configuring your product to allow end users to install your product multiple times in each context on a target system, the **Setup.exe** setup launcher must be used if you want the instance selection dialog to be displayed when appropriate. The **Setup.exe** file displays the instance selection dialog.

To learn more about multiple-instance support, see Installing Multiple Instances of Products.

Billboards and Setup.exe

Basic MSI projects that include billboards require **Setup.exe** because **Setup.exe** displays the billboards at run time.

For more information about billboards, see Displaying Billboards in Basic MSI Installations.

Customizing File Properties for the Setup Launcher

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

Note that project-specific differences are noted where applicable.

InstallShield lets you use custom information for the version resources of the **Setup.exe** setup launcher. The information is displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties.

Settings in InstallShield that Let You Configure Setup.exe Properties

The following table lists various properties that Windows includes on the Properties dialog box, along with the corresponding settings in InstallShield that you can use to configure them.

Note *• The Properties dialog box is different on different versions of Windows. For example, on Windows 7 systems, the version resource information is displayed on the Details tab of the Properties dialog box. However, on Windows XP systems, the version resource information is displayed on the Version tab of that dialog box.*

Also note that some versions of Windows do not show some settings on the Properties dialog box.

Table 4-23 • Source of Information for Setup.exe Properties

Setup.exe Property	InstallShield Setting that Configures the Setup.exe Property
Company Name	The setting that contains the value for this field varies, depending on the project type that you are using. • In Basic MSI and InstallScript MSI projects: Publisher setting in the General Information view. • In InstallScript projects: Company Name setting in the General Information view. • In Advanced UI and Suite/Advanced UI projects: InstallShield uses either a custom value that you provide, or a default InstallShield-specific name. If you want InstallShield to use your own custom value for the company field in an Advanced UI or Suite/Advanced UI project: **1.** In the **Releases** view, select the release that you want to configure. **2.** On the **Setup.exe** tab, in the **Use Custom Version Properties** setting, select **Yes**. **3.** In the **Company** setting, enter the text that you want to use for the company field in the properties of your **Setup.exe** file. If you select No for the Use Custom Version Properties setting or you leave the Company setting blank, InstallShield uses the Publisher setting in the General Information view. If that setting is blank, InstallShield uses the default InstallShield-specific name.
Product Name	The setting that contains the value for this field varies, depending on the project type that you are using. • In Basic MSI and InstallScript MSI projects: Product Name setting for the product configuration in the Releases view. If that setting is blank, InstallShield uses the Product Name setting in the General Information view. • In Advanced UI, InstallScript, and Suite/Advanced UI projects: Product Name setting in the General Information view.

Table 4-23 • Source of Information for Setup.exe Properties (cont.)

Setup.exe Property	InstallShield Setting that Configures the Setup.exe Property
Product Version and File Version	The setting that contains the value for these fields varies, depending on the project type that you are using. • In Basic MSI and InstallScript MSI projects: Product Version setting for the product configuration in the Releases view. If that setting is blank, InstallShield uses the Product Version setting in the General Information view. • In InstallScript projects: Product Version setting in the General Information view. • In Advanced UI and Suite/Advanced UI projects: Product Version setting in the General Information view. You can use the Releases view to override this value. Note that the file version always contains four fields. If you specify fewer than four fields for your product version, the remaining fields are filled with zeros. For example, if you specify a product version of **1.1**, the file version that is used in the version resources of **Setup.exe** is **1.1.0.0**. If you want to override the value in the General Information view in an Advanced UI or Suite/Advanced UI project: **1.** In the **Releases** view, select the release that you want to configure. **2.** On the **Setup.exe** tab, in the **Use Custom Version Properties** setting, select **Yes**. **3.** In the **Version** setting, enter the value that you want to use for the version fields in the properties of your **Setup.exe** file. If you select No for the Use Custom Version Properties setting or you leave the Version setting blank, InstallShield uses the Version setting in the General Information view. If that setting is blank, InstallShield uses the default InstallShield-specific values.
Copyright	InstallShield uses either a custom value that you provide, or the default InstallShield copyright notice. If you want InstallShield to use your own custom value for the copyright field: **1.** In the **Releases** view, select the release that you want to configure. **2.** On the **Setup.exe** tab, in the **Use Custom Version Properties** setting, select **Yes**. **3.** In the **Launcher Copyright** setting, enter the text that you want to use for the copyright field in the properties of your **Setup.exe** file. If you select No for the Use Custom Version Properties setting or you leave the Launcher Copyright setting blank, InstallShield uses the default InstallShield copyright notice.

Table 4-23 • Source of Information for Setup.exe Properties (cont.)

Setup.exe Property	InstallShield Setting that Configures the Setup.exe Property
File Description	InstallShield uses either a custom value that you provide, or the default InstallShield **Setup.exe** description. If you want InstallShield to use your own custom value for the description field: **1.** In the **Releases** view, select the release that you want to configure. **2.** On the **Setup.exe** tab, in the **Use Custom Version Properties** setting, select **Yes**. **3.** In the **File Description** setting, enter the text that you want to use for the description field in the properties of your **Setup.exe** file.  ***Project*** • *In an Advanced UI, InstallScript, or Suite/Advanced UI project—If you select No for the Use Custom Version Properties setting or you leave the File Description setting blank, InstallShield uses the default InstallShield* **Setup.exe** *description.* *In a Basic MSI or InstallScript MSI project—If you select No for the Use Custom Version Properties setting, InstallShield uses the default InstallShield* **Setup.exe** *description. If you select Yes but leave the File Description setting blank, InstallShield uses the value for the Comments setting of the product configuration in the Releases view. If that setting is blank, InstallShield uses the Summary Information Stream Comments setting in the General Information view. If that setting is blank, InstallShield uses the default InstallShield* **Setup.exe** *description.*
Language	For this field, InstallShield uses the language that is specified in the Default Language setting for the release in the Releases view that you are building.

Sample Setup.exe Properties Dialog Boxes

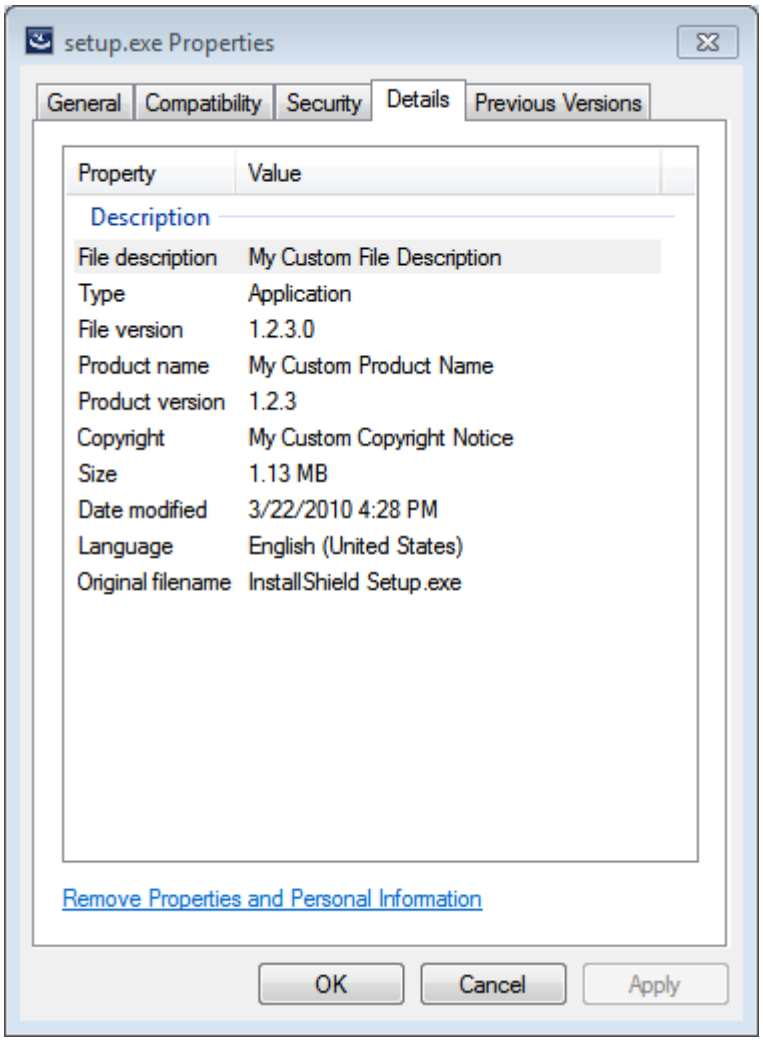


Figure 4-1: Sample Properties for Setup.exe on a Windows 7 Machine

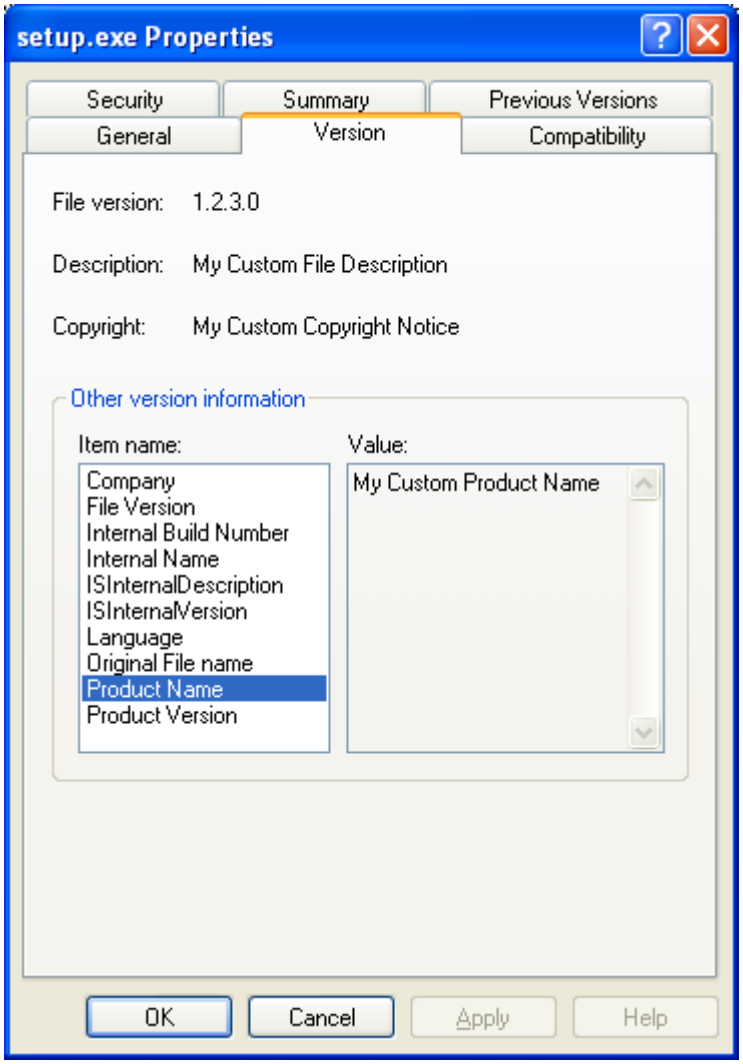


Figure 4-2: Sample Properties for Setup.exe on a Windows XP Machine

Specifying the Icon for the Setup Launcher

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

InstallShield lets you specify the icon that should be used for your **Setup.exe** setup launcher. The icon can be in an .exe, .dll, or .ico file.

End users can see this icon when they view your **Setup.exe** file in Windows Explorer. The icon is also displayed on the Properties dialog box for **Setup.exe**; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties.

If you do not specify an icon, InstallShield uses a default icon for your **Setup.exe** file.



Task ***To specify the icon for your Setup.exe file:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the release that you want to configure.
3. Click the **Setup.exe** tab.
4. In the **Setup.exe Icon File** setting, specify the fully qualified name of the file that contains the icon that InstallShield should use when it creates the **Setup.exe** file at build time.

   To specify a file, type an explicit path or path variable, or click the ellipsis button (**...**) to open the **Change Icon** dialog box, in which you can click the **Browse** button to select a file.

   By default, the icon with index 0 is used; to specify a different icon, either select an icon in the **Change Icon** dialog box or append the icon's index or resource ID (preceded by a minus sign) to the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100.

The next time that you build the release, InstallShield uses the icon that you specified for your **Setup.exe** file.

Distributing Installations

Once you have created your installation, you may need to distribute it to a specified location. This location can be a network drive, a CD, a different location on a local drive, or an FTP site. When you distribute your installation, the disk image created when you built your installation is copied to the location that you specify.

Distributing Releases to a Folder or FTP Site Automatically

When your release is built and tested, the only remaining task is to distribute it to the appropriate location. You can manually copy your release to the appropriate location, or you can use the Events tab in the Releases view to configure the release so that InstallShield automatically copies the release to the appropriate location—a local or network location, or an FTP site.



Task ***To configure InstallShield to automatically distribute your release to a particular location:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the release that you want to configure.
3. Click the **Events** tab.
4. Configure the settings as appropriate. To learn more about the settings on the **Events** tab, see Events Tab for a Release.

***Note** • If your installation consists of only one disk, the contents of the Disk1 folder are copied to the release location, but not the folder itself. If your installation spans across multiple disks, the folders and their contents are copied to the release location.*

***Project** • For InstallScript and InstallScript Object projects, InstallShield automatically copies the release to the location that you specify on the Events tab every time that you build the release.*

For any of the following project types, InstallShield copies all of the relevant files for your release to the specified location whenever you right-click the release in the Releases view and then click Distribute:

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

If you specify a folder and an FTP site on the Events tab, InstallShield copies the files to only the FTP location.

If you want the build engine to copy your release to the specified location after every build in a Basic MSI, InstallScript MSI, or Merge Module project, select Yes for the Distribute After Build setting.

Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand

***Edition** • The ability to distribute releases to virtual machines is available in the Premier edition of InstallShield.*

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

You can configure releases in your projects so that after each successful build of your installation, InstallShield automatically reverts a virtual machine (VM) to a designated snapshot (if one is specified), powers on the VM, and copies your installation to the VM to make it available for testing. You can also alternately perform these testing preparation steps on demand at any time. The testing preparation capability makes it possible to reduce testing time and eliminate manual steps.

The VM can be on a Microsoft Hyper-V Server, a VMware ESX or ESXi Server, or a VMware Workstation.

Requirements for Distributing to VMs

In order to deploy to one of the supported VMs, InstallShield needs to communicate with the virtualization technology that you are using. If you are using VMware virtualization technology (VMware ESX or ESXi Server or VMware Workstation), the VMware VIX API needs to be installed on the same machine as InstallShield. You can do this by either installing VMware Workstation on that machine or by downloading and installing the VMware VIX API from http://www.vmware.com/support/developer/vix-api.

If you are using VMware Workstation, it is recommended that you install VMware Workstation on the same machine as InstallShield so that InstallShield uses the version of the VIX API that was designed for that specific version of VMware Workstation. Although it is likely that newer versions of the VIX API will also work, it seems that the best approach is for InstallShield to use the version of the VIX API that was bundled with your version of VMware Workstation.

Configuring VM-Related Settings



Task ***To configure the VM-related settings for a release to enable distribution to the VM at build time or on demand:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the release that you want to configure.
3. Click the **Events** tab.
4. Expand the **Virtual Machine** area.
5. Use the **Configuration** setting to select the name of the group of VM configuration settings that you want to use when distributing the selected release to a VM:
   - To create a new VM configuration, click the ellipsis button (...) in this setting.
   - To edit an existing VM configuration, select it in this setting and then click the ellipsis button (...).

   In both cases, the **Edit Virtual Machine Configurations** dialog box opens, enabling you to configure machine-wide VM details. For information on using this dialog box, see Edit Virtual Machine Configurations Dialog Box.

   Note that when you specify VM configuration settings in the Edit Virtual Machine Configurations dialog box, InstallShield writes the data that you specify to a file called **VMConfigurations.xml**. This **VMConfigurations.xml** file is a machine-wide file that you can configure once and then use it in other InstallShield projects, as well as share it with other team members. You can also use this file on your build machine. To learn more, see Sharing Virtual Machine Settings for Distribution of Releases.
6. Configure the remaining settings in the **Virtual Machine** area as needed.

   If you configure VM details in these remaining settings, you are overriding some of the machine-wide values that were entered on the Edit Virtual Machine Configurations dialog box.

   For details about all of the settings in the Virtual Machine area, see Events Tab for a Release.



Tip *• If you want InstallShield to distribute the release to the specified VM after each successful build, select Yes for the Distribute After Build setting.*

To perform the staging steps for a built release on demand, right-click the release and then click Distribute to VM.

Preparing Installations for Internet Distribution

The way in which consumers receive software is rapidly changing. Before the advances in Internet technology and the introduction of high-speed connections, all software was shipped on some type of removable medium, such as floppy disks or CD-ROMs. Today, many people download their software directly from the Internet. In order to take advantage of this time- and money-saving software distribution process, you must package your installation in an easily downloadable and installable manner.

There are several criteria that your Web-ready installation may need to meet:

Compressed Size

Although many people now connect to the Internet through high-speed cable modems or DSL lines, others still use lower-speed modems. Package size is very important to people with slower connections due to the increased amount of online time required to download an application.

Self-Extracting

Many file compression utilities require a special client-side application to decompress the application files. This need for another utility complicates the download and installation process for end users. To simplify the installation process, the compression utility that you use should be self-extracting so that no other application is required.

Digitally Signed

To make your customers feel more secure about downloading and installing your software, you can digitally sign your application package. The digital signature identifies you or your company to end users and assures them that the application code has not been altered or tampered with since publication. To learn how to digitally sign your application, see Digital Signing and Security.

Easy to Use

Perhaps the most important aspect of packaging your installation for Internet distribution is making it easy to use. Your end users may not want to specify a location where the installation files should be saved and then search their computer to locate those files. Instead, the installation should be seamlessly integrated into the compression package, requiring only one step to begin the installation.

Proxy Server Support

You may want to configure your installation to download certain files only if they are needed on the target system. For example, the Windows Installer engine, the .NET Framework, and some InstallShield prerequisites may already be present on some or most target systems. Instead of embedding these files in your installation (which would increase your overall installation size), you can configure your project so that only the ones that are needed are downloaded at run time.

If your end users access the Internet through a proxy server and your installation is configured to download files, the installation uses the system proxy settings that are manually configured in Internet Explorer during the download. This occurs even if another browser on the target system is the default browser.

Note that InstallShield does not include support for the Automatically Detect Settings functionality in Internet Explorer. (If end users have the Automatically Detect Settings check box selected in Internet Explorer for their LAN connection and the installation needs to download files, the installation fails because the files cannot be downloaded. If it is possible that your end users may have the Automatically Detect Settings check box selected in Internet Explorer for their LAN connections, you may want to embed all of the files in your installation rather than

configure them to be downloaded; if the files are embedded, the failures can be avoided.) However, InstallShield does support the Automatic Configuration Script functionality that is set up for LAN connections in Internet Explorer.

Redistributable Files that Are Distributed with an InstallScript Installation



Project • *This information applies to InstallScript projects.*

The following files are automatically created by InstallShield and can be redistributed with your software distribution, as discussed in your End-User License Agreement.The media build automatically includes them as needed in your disk image folders.

Main Media Files

- data<2, 3, 4, ...>.cab
- data1.cab
- data1.hdr
- ISSetup.dll
- layout.bin
- setup.exe
- setup.ini
- setup.inx

Optional Files

- setup.isn—Skin File
- setup.htm—One Click Install .htm page

Deploying UWP App Packages



Edition • *UWP app package creation is available in Basic MSI projects.*



Important • *The Windows 10 Anniversary Update is required for installing and testing a UWP app package (.appx) with desktop extensions (Desktop Bridge) included. To digitally sign the UWP app package, InstallShield must be installed on a Windows 10 machine or a machine with the Windows 10 SDK installed.*



Tip • *Sideloading must be enabled for a UWP app package to be successfully sideloaded. For more information about sideloading, refer to the Enable your device for deployment MSDN article.*

The following table provides an overview of the ways you can distribute UWP app packages:

Table 4-24 • UWP App Distribution Methods

Distribution Method	Description	Distribution Method Setting	Signing Considerations
Windows Store	Upload the UWP app package to the Windows Store. Refer to the Packaging UWP apps (MSDN article) for more information. Any package distributed through the Windows Store is subject to additional policies that limit its contents.	Windows Store	This type of app doesn't need to be signed before it is uploaded to Microsoft because Microsoft signs the package.
Sideloaded and added to a Suite/ Advanced UI Project	Add a sideloading UWP app package to a Suite/Advanced UI project as described in Adding a Sideloading UWP App Package (.appx) to a Suite/Advanced UI Project. A common example of a sideloading app is one that is distributed to an enterprise environment and is internal to a company only. You can use condition checks such as the **UWP App Package Eligible** eligibility condition of a UWP app package (.appx) or the **UWP Type Present** condition in either an exit condition or a package's eligibility condition as well as other condition checks to employ this type of distribution method. This type of scenario provides flexibility to let end users conditionally install an .msi package versus a UWP app package. For more information, refer to the Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects topic.	Direct Distribution (Sideload)	If a package will be sideloaded **and** added to a Suite, the UWP app package must be signed before it is built into the Suite/Advanced UI project.
Sideloaded and delivered as a UWP app package (.appx)	Deliver a UWP app package (appx) file to your customer as an executable installer that can be double-clicked to perform the installation. A common example of a sideloading app is one that is distributed to an enterprise environment and is internal to a company only.	Direct Distribution (Sideload)	If a package will be sideloaded, you will typically need to sign it, unless your customer wishes to sign it themselves. The package must be signed before the double-click installation will succeed.

Creating Transforms



Project • *This information applies to Transform projects.*

A transform (.mst file) is a simplified Windows Installer database that contains the differences between two .msi databases. Transforms enable an administrator to apply modified settings to a database when deploying an installation package.

Customizing Installations

For example, a user may need to customize an installation in different ways for different departments in their company. The traditional office suite comes with a spreadsheet program, a word processor, and a presentation tool. The accounting department may need only the spreadsheet and the presentation programs, while the writing department may need the word processor and the spreadsheet. A third department may need the entire suite of applications.

Rather than manually setting up every person in the company, a user can take the original installation of the entire suite, customize it to the needs of each department, and then create a transform between the two packages. A transform needs to be created for each separate product configuration.

Creating Transforms

InstallShield makes the task of creating multiple configurations of your product as easy as stepping through a wizard. You can create a transform by using either of the two methods described below.

Creating a Transform by Comparing Two .msi Packages

In one method of creating a transform, a base .msi package serves as the starting point for all of your customization, and the target package contains all of the changes that you would like to be reflected in the installation. Once you have prepared the base .msi package and the target .msi package, you can use the Transform Wizard to create a transform. When you finish this wizard, InstallShield compares the two packages and creates a transform (.mst file) that contains the differences between the two packages.

Creating a Transform Based on a Single .msi Package

Another way to create a transform is to create a new transform project. When you create a new transform project, the Open Transform Wizard launches. Use the Open Transform Wizard to open an existing .msi package in Direct MST mode and edit the project as needed. InstallShield evaluates the differences between the base .msi package and the changes you make in Direct MST mode and creates a transform project (.mst file) that contains the differences.

This method of creating a transform enables you to specify additional transforms that should be incorporated into your transform project. It also lets you specify default user-interface responses for end users.

Applying Transforms

Once a transform has been created, it can be applied at run time, depending on whose machine the application is being installed. For example, you can check if the target machine is in the accounting department. If it is, the accounting transform is applied to the original installation, and only the spreadsheet and presentation programs are installed.

Creating a Transform by Comparing Two .msi Packages

Project *• This information applies to Transform projects.*

Note *• If a transform project includes new files (files not found in the original .msi file), these files will not be found when running the installation. To avoid this issue, you can either put the transform project in the same location as the base .msi package, or manually copy the source files to the location of the base .msi package.*

Task ***To create a transform by comparing two .msi packages:***

1. Open the base .msi package in InstallShield.
2. Save the project with a new name. This new project is your target project. For more information, see Saving a Project with a New Name and Location.
3. Make all of the changes that you would like to be reflected in the installation and then build a release.
4. On the **Tools** menu, click **Create/Apply Transform**. The Transform Wizard opens.
5. Complete the panels in the **Transform Wizard** to create the transform.

Creating a Transform Based on a Single .msi Package

Project *• This information applies to Transform projects.*

Note *• If a transform project includes new files (files not found in the original .msi file), these files will not be found when running the installation. To avoid this issue, you can either put the transform project in the same location as the base .msi package, or manually copy the source files to the location of the base .msi package.*

Task ***To create a transform based on a single .msi package:***

1. On the **File** menu, click **New**. The **New Project** dialog box opens.
2. On the **Windows Installer** tab, click **Transform**.
3. In the **Project Name** box, type a name for your transform project.
4. In the **Location** box, type the path where your transform project should be stored or click **Browse** to find the location.
5. Click **OK**. The **Open Transform Wizard** opens.
6. Complete the panels in the **Open Transform Wizard** to create the transform.

Applying Transforms

Project *• This information applies to Transform projects.*

When you apply a transform, you are modifying a base .msi package to incorporate changes that you included in the transform (an .mst file).

The two methods of applying a transform are described below.

Applying a Transform from the Command Line

One way to apply a transform is to pass a command line to Msiexec.exe or Setup.exe. When you apply a transform from the command line, you are changing the database of your .msi package at run time. To apply a transform at run time, your command-line statement should look something like this, specifying the desired transform with the TRANSFORMS property:

```
msiexec /i "ProductName.msi" TRANSFORMS="YourTransform.mst"
```

Use semicolons to separate multiple transforms.

Caution *• Do not use semicolons in the name of your transform because the Windows Installer service interprets semicolons as file name separators.*

When you specify more than one transform at the command line, the transforms are applied in the order specified. For example, you could have a base package that has only one feature, and one transform that adds a second feature, and another transform that changes the second feature in some way. The transform that adds the feature must be specified before the transform that changes the second feature; otherwise, the change will not be made properly.

Using the Transform Wizard to Apply a Transform

When you use the Transform Wizard to apply a transform, you create an installation package (.msi package) that reflects all of the changes contained in the transform file. This can be useful for network administrators who are customizing applications for all of their users.

Task ***To launch the Transform Wizard:***

On the **Tools** menu, click **Create/Apply**.

Editing Transforms

Project *• This information applies to Transform projects.*

InstallShield enables you to edit .msi packages without having to convert them to an InstallShield (.ism) project. You can save the changes made as a transform (.mst) file.

Opening Existing Transforms for Editing

In InstallShield, you can browse for an existing .mst file. Before InstallShield can open the .mst file, it needs the base .msi to which this file should be applied. Also, it requires the names of any additional .mst files that should be applied to the base .msi file before opening the .mst file for editing.

The first time that you open an .mst file in InstallShield, the Open Transform Wizard launches. This wizard enables you to specify the .mst file, the base .msi file, and any additional transforms to apply before editing. The last panel of this wizard—the Create a Response Transform panel—lets you specify whether or not your transform should be a response transform. A response transform includes default responses for the user interface portion of the installation. If you specify that you want to create a response transform, the user interface elements of the installation are executed, enabling you to customize the options available in each panel of the installation wizard.

When the Open Transform Wizard is finished, the transforms are applied and the .mst file opens in InstallShield in Direct MST mode (which is similar to Direct Edit mode). The data specified in the wizard is saved so that the next time that you open the .mst file, the transform project opens directly in InstallShield.

Task ***To open an existing transform:***

1. On the **File** menu, click **Open**. The **Open** dialog opens.
2. Select the .mst file that you would like to open.
3. Click **Open**. The **Open Transform Wizard** opens.

Saving .msi Files as Transforms

When you are editing an .msi file in Direct Edit mode, you can save the file as an .mst file.

Task ***To save an .msi file as an .mst file:***

1. On the **File** menu, click **Save As**. The **Save As** dialog box opens.
2. In the **Save as type** box, select **Windows Installer Transforms (*.mst)**.

All the changes made since the last save are saved to the .mst file specified.

Saving Transforms as .msi Files

You can save an .mst file as an .msi file.

Task ***To save a transform as an .msi file:***

1. Open the transform project in InstallShield if you have not already done so.
2. On the **File** menu, click **Save As**. The **Save As** dialog box opens.
3. In the **Save as type** box, select **Windows Installer Packages (*.msi)**.

An .msi file is created. It contains all of the data from the base .msi file, the .mst file that you were editing, and any additional .mst files applied. The project is treated as a Direct Mode MSI project. For more information, see Editing .msi and .msm Databases in Direct Edit Mode.

Response Transforms

Project • *This information applies to Transform projects.*

A response transform is a type of transform that enables administrators to set up default values for an installation by simply running the user interface sequence and entering the desired values. For example, you might want to create a response transform that contains default values for which features are installed, for the installation location of the application, and for the company name. The default values that you specify are used as a starting point for your transform.

To create a response transform, use the Open Transform Wizard.

Creating a Response Transform

Project • *This information applies to Transform projects.*

Task ***To create a response transform:***

1. Open the Open Transform Wizard.
2. Complete each panel of the wizard as needed. On the Create Response Transform panel, select **Create response transform**.
3. In the **Command line properties (optional)** box, specify command-line properties (in property name/value pairs separated by semicolons) that you would like to be passed to the response transform.
4. Click **Finish**. The user interface elements of the installation are executed.
5. Complete the simulated installation, customizing the options available in each panel of the installation wizard as needed. No file transfer takes place, and no changes to your machine occur.
6. When you reach the end of the installation sequence, click **Install**.

The simulated installation exits, and InstallShield saves all of the changes that you made during the simulated installation as values in the Property table of your transform project. These values are used as default values in the your installation.

Modifying a Response Transform

Project • *This information applies to Transform projects.*

Task ***To modify an individual response in an existing transform project:***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. Select the transform project (.mst file) whose responses you would like to modify.
3. Click **Open**. InstallShield opens your transform project in Direct MST mode.
4. In the **Direct Editor**, click the **Property** table.
5. Modify the values of any response properties as needed. Examples of properties that you may want to modify include **COMPANYNAME** and **USERNAME**.

As an alternative, you can also follow the same steps described for Creating Transforms; these steps enable you to launch the user interface of the installation and select the appropriate responses.

Testing a Response Transform

Project • *This information applies to Transform projects.*

To see the changes you made to the responses, you can test the user interface part of your transform by doing any of the following:

- On the **Build** menu, click **Test**.
- Click the **Test** button.
- Press CTRL+T.

Configuring Server Locations

Project • *This information applies to Transform projects.*

If end users install a product from a network server or Web site, and they install features to run from the server or Web site, the product may need access to the .msi package and related files on the server or Web site sometime after the initial installation. The product may also require access if a file is deleted or becomes corrupted; the installation can copy the problematic file or files automatically from the server or Web site.

InstallShield enables you to specify in your transform a list of network or URL source paths to your product's installation package. The paths in this list are stored in the **SOURCELIST** property. Windows Installer appends this list to the end of an end user's existing source list for the product at run time.

Adding Server Locations

Project • *This information applies to Transform projects.*

Task ***To add a network server path or a Web site URL for a server location:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Server Locations** setting, click the ellipsis button (**...**). The **Server Locations** dialog box opens.
3. Specify server locations as needed.
4. Click **OK**.

InstallShield updates the value in the Server Locations setting with the locations that you specified.

Modifying Server Locations

Project • *This information applies to Transform projects.*

Task ***To modify a location in the server list:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Server Locations** setting, click the ellipsis button (**...**). The **Server Locations** dialog box opens.
3. Modify server locations as needed.
4. Click **OK**.

InstallShield updates the value in the Server Locations setting as required.

Removing Server Locations

Project • *This information applies to Transform projects.*

Task ***To remove a path for a server location:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Server Locations** setting, click the ellipsis button (**...**). The **Server Locations** dialog box opens.
3. In the **Source Paths** list, select the path that you want to remove.
4. Click the **Delete** button.
5. Click **OK**.

Changing the Order of Server Locations

If an installation needs access to the source files that are stored on a network server or a Web site, it uses the first accessible path in the list of server locations. InstallShield lets you change the order of paths if necessary.



Project • *This information applies to Transform projects.*



Task ***To change the order of the server locations:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Server Locations** setting, click the ellipsis button (**...**). The **Server Locations** dialog box opens.
3. In the **Source Paths** list, select the path that you want to move.
4. Click the **Move Up** and **Move Down** buttons as needed to change the order.
5. Click **OK**.

5

Creating Advanced UI and Suite/ Advanced UI Installations

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Advanced UI and Suite/Advanced UI installations are bootstrap applications that package together installations and InstallShield prerequisites as a single installation while providing a unified, fully customizable user interface. They use a setup launcher (**Setup.exe**) to conditionally launch packages on target systems as needed.

The following diagram shows the various parts of a Suite/Advanced UI installation, which has support for running multiple primary packages (including .exe packages, sideloading UWP app packages (.appx), and Windows Installer transactions) and multiple InstallShield prerequisite packages. Note that the parts of an Advanced UI installation are similar, except that an Advanced UI installation can run only one primary package, but multiple InstallShield prerequisite packages.

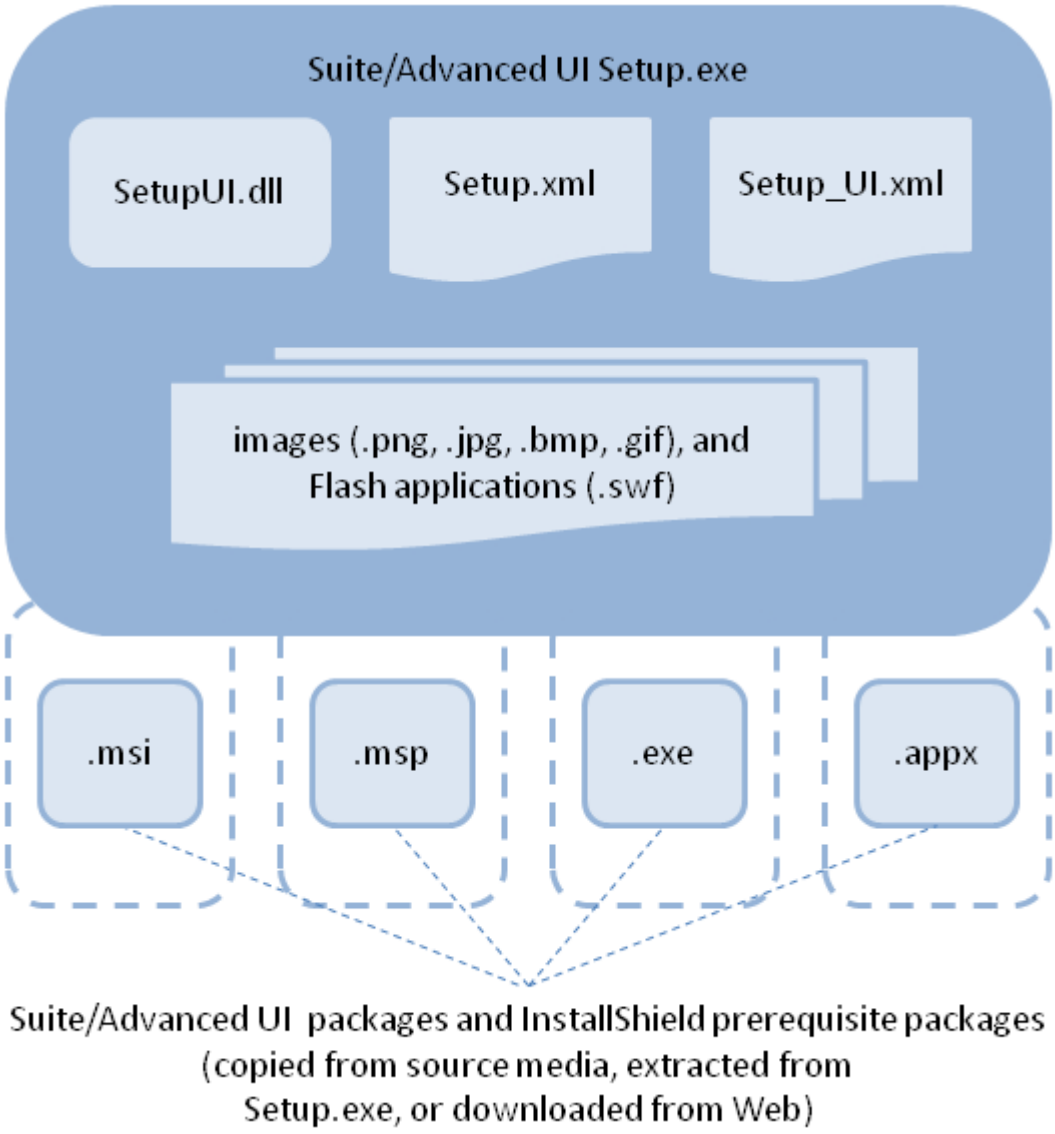


Figure 5-1: Parts of a Suite/Advanced UI Installation

At build time, InstallShield uses the various sections of the .issuite file—the Advanced UI project's or Suite/Advanced UI project's XML project file—to create **Setup.xml** and **Setup_UI.xml**, and bundle these together with the Advanced UI's or Suite/Advanced UI's packages and other files that are required to run the Advanced UI or Suite/Advanced UI installation. The resulting Advanced UI or Suite/Advanced UI installation includes the following files:

- **Setup.exe**, which contains the Advanced UI or Suite/Advanced UI engine
- **Setup.xml**, which describes the packages that are in the Advanced UI or Suite/Advanced UI installation, the conditions under which they are launched, and the string entries that are used in the wizard user interface of the Advanced UI or Suite/Advanced UI installation
- **Setup_UI.xml**, which defines style settings, layout information, and other UI parameters for the wizard user interface
- **SetupUI.dll**, which consists of the Advanced UI and Suite/Advanced UI library that is used to render the UI of Advanced UI and Suite/Advanced UI installations

InstallShield also gathers the packages in the Advanced UI or Suite/Advanced UI project and sets up the folder structure, if applicable, for the packages and their associated uncompressed files. In addition, InstallShield incorporates the support files of the Advanced UI or Suite/Advanced UI installation; these support files are resources such as EULAs and image files that are shown on wizard pages.

When an end user launches the Advanced UI or Suite/Advanced UI installation, the Advanced UI or Suite/Advanced UI engine reads the **Setup.xml** file, loads the **SetupUI.dll** file, evaluates the conditions that are defined for the installation and each of its packages, follows the instructions in the **Setup.xml** file, (if applicable) downloads packages and files that are needed on the target system, runs the packages that need to be run, and cleans up resources.

Note that Advanced UI and Suite/Advanced UI installations require that Windows Installer 3.1 or later be present on target systems. They also require Windows XP or later or Windows Server 2003 or later.

For more information on the Advanced UI and Suite/Advanced UI types of projects, refer to this section of the documentation.

Advanced UI Projects vs. Suite/Advanced UI Projects



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield.*

Advanced UI and Suite/Advanced UI projects enable you to develop bootstrap applications for installations and create a unified, fully customizable user interface. The following table compares various capabilities and traits of these two types of projects.

Table 5-1 • Summary of Differences Between Advanced UI Projects and Suite/Advanced UI Projects

Comparison Category	Advanced UI Projects	Suite/Advanced UI Projects
Edition of InstallShield For more details on the differences between the Premier and Professional editions, see Upgrading from Other InstallShield Editions.	InstallShield Professional	InstallShield Premier
Project file extension	.issuite	.issuite

Table 5-1 • Summary of Differences Between Advanced UI Projects and Suite/Advanced UI Projects (cont.)

Comparison Category	Advanced UI Projects	Suite/Advanced UI Projects
Supported file format for primary packages For information on selecting the appropriate package file format, see Guidelines for Adding Packages to an Advanced UI or Suite/Advanced UI Project.	.msi package, .msp patch, InstallScript package	.msi package, .msp patch, InstallScript package, .exe package, .appx package, Web Deploy package (.zip), Windows Installer transaction You can also include Basic MSI and InstallScript projects (.ism) as packages in a Suite/Advanced UI installation. At build time, InstallShield first builds the specified releases in the InstallShield projects, and includes them as packages in the Suite/Advanced UI installation that it generates. Note that the ability to create and build a Suite/Advanced UI installation that includes a sideloading UWP app package requires Windows Vista or later or Windows Server 2008 or later on the machine that has InstallShield or the Standalone Build.
Number of primary packages that you can include in the project For more detailed, see Primary Packages vs. Dependency Packages in Advanced UI and Suite/Advanced UI Projects.	1 This capability enables you to create a contemporary, customizable user interface for a single package.	1 or more This capability enables you to create a contemporary, customizable user interface for one or more packages. This capability also lets you package together multiple separate installations as a single installation while providing a unified user interface.
Support for including InstallShield prerequisites in the project To learn more, see Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project.	Include as many InstallShield prerequisites as needed as .msi packages, .msp packages, and .exe packages, depending on the type of file that is configured to run for the prerequisite.	Include as many InstallShield prerequisites as needed as .msi packages, .msp packages, and .exe packages, depending on the type of file that is configured to run for the prerequisite.

Table 5-1 • Summary of Differences Between Advanced UI Projects and Suite/Advanced UI Projects (cont.)

Comparison Category	Advanced UI Projects	Suite/Advanced UI Projects
Support for indicating which packages are primary packages and which are secondary packages To learn more, see Primary Packages vs. Dependency Packages in Advanced UI and Suite/Advanced UI Projects.	Limited flexibility. You can add one .msi, .msp, or InstallScript package as a primary package to your project. Any InstallShield prerequisites that you include in your project are added as dependency packages. This cannot be changed.	Full flexibility. You can add any of the supported package types to your project and easily specify whether they are primary or dependency packages. Any InstallShield prerequisites that you include in your project are added as dependency packages, but you can override this and mark them as primary packages.
Support for creating actions and scheduling them to run during the appropriate run-time events To find out more, see Using Actions to Extend the Behavior of a Suite/Advanced UI Installation.	Not available.	A Suite/Advanced UI project includes an Events view that lets you define actions that you want to occur at run time on target systems. You can schedule an action to occur during a particular event in the Events view, or you can assign to a package in the Packages view. In addition, you can schedule an action to occur when a wizard page or secondary window is opened or closed, or when an end user uses a control (for example, clicks a button).
Support for build events For more information, see Specifying Commands that Run Before, During, and After Builds.	Not available.	A Suite/Advanced UI project includes release settings that let you schedule commands that run before InstallShield starts building the release, as well as after InstallShield has built and signed the release.
Run-time language support For more information, see Run-Time Language Support in InstallShield.	The end-user interface of an Advanced UI installation is limited to a single language that you determine when installing InstallShield.	Create a single installation that displays end-user text in multiple languages. Change wizard pages and messages to any one of 34 additional languages using pre-translated strings.

Table 5-1 • Summary of Differences Between Advanced UI Projects and Suite/Advanced UI Projects (cont.)

Comparison Category	Advanced UI Projects	Suite/Advanced UI Projects
Support for distributing your installation to virtual machines that InstallShield provisions at build time To learn more, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.	Not available.	After each successful build of your installation, InstallShield automatically reverts a virtual machine (VM) to a designated snapshot, powers on the VM, and copies your Suite/Advanced UI installation to the VM to make it available for testing. This capability makes it possible to reduce testing time and eliminate manual steps.

Organizing Features, Packages, Prerequisites, and Files in an Advanced UI or Suite/Advanced UI Installation



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

An Advanced UI or Suite/Advanced UI installation is a bootstrap application whose primary purpose is typically to conditionally launch packages on target systems as needed. An Advanced UI or Suite/Advanced UI installation may also need to run InstallShield prerequisites to install third-party or custom redistributables to ensure that the product can function properly.

A feature is the smallest installable part of an Advanced UI or Suite/Advanced UI installation from the end user's perspective. Each package and InstallShield prerequisite that is included in an Advanced UI or Suite/Advanced UI installation should be associated with a feature in the project.

For information on adding and configuring packages and InstallShield prerequisites in an Advanced UI or Suite/Advanced UI project, review this section of the documentation.

Guidelines for Adding Packages to an Advanced UI or Suite/Advanced UI Project



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

File Format Recommendations

The recommended file format for a package in an Advanced UI or Suite/Advanced UI project is one that does not include a bootstrap application or setup launcher (**Setup.exe**). The following package types are recommended for Advanced UI and Suite/Advanced UI projects:

- .msi package
- .msp patch
- InstallScript package—The package must be an uncompressed InstallScript installation that is built in InstallShield 2012 Spring or later.

To learn more about use of these types of packages in an Advanced UI or Suite/Advanced UI project, see the following:

- Adding an .msi Package, an .msp Patch, or a Transaction to an Advanced UI or Suite/Advanced UI Project
- Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project

Suite/Advanced UI installations also have support for deploying Web Deploy packages (.zip) to IIS Web servers and the cloud. For more details, see Adding a Web Deploy Package to a Suite/Advanced UI Project.

In addition, InstallShield includes support for adding sideloading UWP app packages (.appx) to Suite/Advanced UI projects. Sideloading an app is the process of installing an app without obtaining it through the Windows Store. For more information, see Adding a Sideloading UWP App Package (.appx) to a Suite/Advanced UI Project. Because UWP app packages are supported in limited scenarios, these packages are typically used in concert with a more common installation format, such as an .msi package.

InstallShield also enables you to add executable packages (.exe) to a Suite/Advanced UI project. Examples include:

- Setup launcher executable file that was built in InstallShield for a Windows Installer–based installation
- Setup launcher executable file that was built in InstallShield for a Windows Installer–based patch
- Setup launcher file for an InstallScript installation (This includes InstallScript installations that were built in InstallShield 2012 or earlier, as well as compressed InstallScript installations.)
- Setup launcher executable file that was built in a tool other than InstallShield

The .msi, .msp, and InstallScript packages are preferred to the setup launcher executable file formats for various reasons:

- The wizard interface of the Advanced UI and Suite/Advanced UI setup launcher can show the incremental progress of the recommended file formats as they are being run on target systems, but not of the executable packages.
- When you add one of the recommended file formats as a package to an Advanced UI or Suite/Advanced UI project, InstallShield automatically adds an appropriate eligibility condition that prevents the Advanced UI or Suite/Advanced UI installation from allowing an earlier version of a package to install over a later version. If you add an executable package to your Suite/Advanced UI project, you must manually define an eligibility condition that prevents such downgrades.
- If you add one of the recommended file formats as a package to an Advanced UI or Suite/Advanced UI project, you are not required to define for that package a detection condition that evaluates whether the package is already installed on target systems, since the Advanced UI or Suite/Advanced UI installation blocks this scenario from occurring. If you add an executable package to your Suite/Advanced UI project, you must manually define that sort of detection condition for the package.
- The Advanced UI or Suite/Advanced UI setup launcher automatically suppresses the user interface of the recommended file formats, in favor of the wizard interface of the Advanced UI or Suite/Advanced UI installation. If your project includes an executable package, you must manually suppress its user interface and have the Advanced UI or Suite/Advanced UI installation launch it silently. Otherwise, at run time, the installation shows the Advanced UI or Suite/Advanced UI user interface and the separate user interface of the .exe package, presenting two disparate user interfaces.Adding an .msi Package, an .msp Patch, or a Transaction to an Advanced UI or Suite/Advanced UI Project

For more information, see Adding an Executable Package (.exe) to a Suite/Advanced UI Project.

Media Type Requirements

An Advanced UI or Suite/Advanced UI installation cannot install a package that spans multiple disks (for example, CDs or DVDs). It also cannot install a multiple-disk package from a fixed disk such as a network. Therefore, any package (.msi package, .msp package, InstallScript package, .exe package, Web Deploy package, or UWP App package (.appx) that you include in an Advanced UI or Suite/Advanced UI project must be a single-disk removable media type, a network image media type, or a Web media type.

Adding an .msi Package, an .msp Patch, or a Transaction to an Advanced UI or Suite/Advanced UI Project



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where appropriate.

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add the following types of Windows Installer–based installations to Advanced UI and Suite/Advanced UI projects:

- Windows Installer packages (.msi packages)
- Windows Installer patches (.msp packages)

InstallShield also lets you add Windows Installer transactions to Suite/Advanced UI projects (but not to Advanced UI projects).

A transaction consists of Windows Installer packages (.msi) and Windows Installer patches (.msp) that you want to be run on target systems using transaction processing, a feature of Windows Installer 4.5 and later. The packages are chained together and processed as a single transaction. Each Suite/Advanced UI installation can have multiple separate transactions. If one or more of the packages in a transaction cannot be installed successfully or if the end user cancels the installation, Windows Installer initiates rollback for all of the chained packages within the current transaction to restore the system to its earlier state.

Adding an .msi or .msp Package That Runs Without Transaction Processing

Task ***To add an .msi or .msp package that is run without transaction processing to your Advanced UI or Suite/Advanced UI project:***

1. In the View List under **Organization**, click **Packages**.
2. Right-click the **Packages** explorer and then click the appropriate command: **New Windows Installer Package (.msi)** or **New Patch Package (.msp)**. The **Select the installation file for this package** dialog box opens.
3. Select the package file that you want to add to your Advanced UI or Suite/Advanced UI project, and then click **Open**. The **Add Files for This Package** dialog box opens.
4. Specify whether you want to include additional folders and files with your package file, and if so, whether you want to include dynamically linked files.

InstallShield adds the package, plus any additional folders and files, in the Packages explorer.

Adding a Package That Runs With Transaction Processing

Task ***To add a package that is run with transaction processing to your Suite/Advanced UI project:***

1. In the View List under **Organization**, click **Packages**.
2. Right-click the **Packages** explorer, and then click **New Transaction**. InstallShield adds a **Transaction** folder to the **Packages** explorer.
3. In the **Packages** explorer, right-click the **Transaction** folder and then click the appropriate command: **New Windows Installer Package (.msi)** or **New Patch Package (.msp)**. The **Select the installation file for this package** dialog box opens.

4. Select the package file that you want to add to your Suite/Advanced UI project, and then click **Open**. The **Add Files for This Package** dialog box opens.
5. Specify whether you want to include additional folders and files with your package file, and if so, whether you want to include dynamically linked files.

InstallShield adds the package, plus any additional folders and files, in the Packages explorer.

Important • *Running a package in a transaction from a Suite/Advanced UI installation requires Windows Installer 4.5 or later. If a package in a transaction is launched by a Suite/Advanced UI installation on a target system that has an earlier version of Windows Installer, the Suite/Advanced UI installation fails. Therefore, if you are using transaction processing for packages in your Suite/Advanced UI installation, it is recommended that you consider performing one or more of the following tasks:*

- *Include the Windows Installer 4.5 redistributable as a package in the Suite/Advanced UI project for target systems that have an earlier version, and schedule it before the packages that use transaction processing.*
- *Create an exit condition that prevents the entire Suite/Advanced UI installation from running on systems that do not have Windows Installer 4.5 or later. To learn more, see Defining Exit Conditions for an Advanced UI or Suite/Advanced UI Installation.*
- *Create an eligibility condition for each package in the transaction that checks for Windows Installer 4.5 or later and does not launch those packages in the transaction. For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.*

Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between the Suite/Advanced UI and Advanced UI project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Requirements for Adding an InstallScript Installation as an InstallScript Package (Instead of as an Executable Package) to an Advanced UI or Suite/Advanced UI Project

InstallShield lets you add an InstallScript package to an Advanced UI or Suite/Advanced UI project if the InstallScript package meets the following requirements:

- The InstallScript package is uncompressed.
- InstallShield 2012 Spring or later is used to build the InstallScript package and the Advanced UI or Suite/Advanced UI installation.

- The InstallScript package uses an event-based script; it should not use the program...endprogram style script. For information on converting a script that uses a program...endprogram block to an event-based script, see OnShowUI.

If any of these conditions cannot be met, you can include the InstallScript installation as an executable file package (.exe) in your Suite/Advanced UI project, instead of as an InstallScript package. However, including the InstallScript installation as an InstallScript package instead of as an executable package (.exe) is preferred. For more information, see Guidelines for Adding Packages to an Advanced UI or Suite/Advanced UI Project.

Note • *If you include an InstallShield 2012 or earlier InstallScript installation as a package in an Advanced UI or Suite/Advanced UI installation, the Advanced UI or Suite/Advanced UI installation encounters a run-time error when it attempts to launch the InstallScript package.*

Instructions for Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project

Task ***To add an InstallScript package to your Advanced UI or Suite/Advanced UI project:***

1. In the View List under **Organization**, click **Packages**.
2. Right-click the **Packages** explorer and then click **InstallScript Package (.hdr)**. The **Select the installation file for this package** dialog box opens.
3. Select the **data1.hdr** file of the InstallScript installation, and then click **Open**. The **data1.hdr** file must be in the same folder as the remaining files of the uncompressed InstallScript installation that you want to include as a package in your Advanced UI or Suite/Advanced UI project.

Tip • *When you select the* **data1.hdr** *file for the InstallScript installation, InstallShield automatically includes the necessary .cab files, InstallScript files (.rul), and other files that the Advanced UI or Suite/Advanced UI installation needs to launch the InstallScript package in an Advanced UI or Suite/Advanced UI installation.*

InstallShield adds the package, plus any additional folders and files, in the Packages explorer.

Tip • *To learn more about the inclusion of InstallScript packages in Advanced UI and Suite/Advanced UI projects, see Special Considerations for InstallScript Packages in Advanced UI and Suite/Advanced UI Projects.*

Special Considerations for InstallScript Packages in Advanced UI and Suite/Advanced UI Projects

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between the Suite/Advanced UI and Advanced UI project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Note the following details if you are including an InstallScript package in your Advanced UI or Suite/Advanced UI project, or if you are including an InstallScript project (.ism) in your Suite/Advanced UI project.

Run-Time User Interface, Events, and Functions

When an Advanced UI or Suite/Advanced UI installation launches an InstallScript package, the Advanced UI or Suite/Advanced UI installation displays its own user interface (UI) while automatically suppressing the UI of the InstallScript package. The Advanced UI or Suite/Advanced UI installation also displays progress information for the InstallScript package. To make these changes possible, the Advanced UI or Suite/Advanced UI installation uses several Advanced UI– and Suite/Advanced UI–specific InstallScript events and functions by default, and ignores some of the standard InstallScript events and functions. The Advanced UI– and Suite/Advanced UI–specific InstallScript events also augment how file transfer starts.

In a standard InstallScript installation that is launched via the InstallScript **Setup.exe** file (that is, not launched from an Suite/Advanced UI installation), most events are called directly from the OnShowUI event. In an Advanced UI or Suite/Advanced UI installation that launches an InstallScript package, OnShowUI is replaced with OnSuiteShowUI. For more information, including the Advanced UI– and Suite/Advanced UI–specific events that are launched by default through OnSuiteShowUI, see OnSuiteShowUI.

The InstallScript language includes some Advanced UI– and Suite/Advanced UI–specific functions that enable interaction between an InstallScript package and the Advanced UI or Suite/Advanced UI installation that is running it. For more information, see Advanced UI and Suite/Advanced UI Interaction Functions.

Note the following guidelines:

- Because an Advanced UI or Suite/Advanced UI installation suppresses the standard InstallScript UI of an InstallScript package, any script dialog calls (for example, **DefineDialog**, **EzDefineDialog**) return an error. Therefore, no dialog calls should be made in the Advanced UI– or Suite/Advanced UI–specific events or in feature transfer events.
- Advanced UI and Suite/Advanced UI installations route all script-provided message box functionality (for example, **MessageBox**, **SprintfBox**) through the SuiteErrorReport function. Calls to these functions should not be made as a way of requesting user input except in error cases. All user input should be obtained through the Advanced UI or Suite/Advanced UI wizard pages, which you can configure in the Wizard Interface view of an Advanced UI or Suite/Advanced UI project. This enables you to provide a consistent UI experience for all of the packages in an Advanced UI or Suite/Advanced UI installation.
- Non-InstallScript UI (such as custom UI implemented through DLLs or other means) should not be called from Advanced UI– or Suite/Advanced UI–specific InstallScript events or feature transfer events. All user input should be obtained through the Advanced UI installation's or Suite/Advanced UI installation's UI.

Passing Data from an Advanced UI or Suite/Advanced UI Installation to an InstallScript Package in the Advanced UI or Suite/Advanced UI Installation

To pass data from an Advanced UI or Suite/Advanced UI installation to an InstallScript package through the command line, use the CMDLINE variable. For more information, see CMDLINE.

To pass Advanced UI or Suite/Advanced UI property values to an InstallScript package, use the InstallScript functions SuiteGetProperty or SuiteFormatString.

Run-Time Detection for Launching Through an Advanced UI or Suite/Advanced UI Installation

To determine whether the InstallScript installation is running as an InstallScript package in an Advanced UI or Suite/Advanced UI installation, use the SUITE_HOSTED variable in your InstallScript code. For more information, see SUITE_HOSTED.

Adding an InstallShield Project (.ism) as a Package in a Suite/Advanced UI Project



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add Basic MSI and InstallScript project files (.ism) as packages to a Suite/Advanced UI project. When you start a build of a release for a Suite/Advanced UI project that includes one or more InstallShield project packages, InstallShield first builds the designated releases in the associated InstallShield projects, and includes them as packages in the Suite/Advanced UI installation that it generates. InstallShield adds the output of the Basic MSI and InstallScript projects to the Suite/Advanced UI installation as .msi and .hdr packages.

Note that to include a Basic MSI or InstallScript project in your Suite/Advanced UI project, the Basic MSI or InstallScript package must have been saved in the same version of InstallShield as the Suite/Advanced UI project.



Task ***To add a Basic MSI project or an InstallScript project to your Suite/Advanced UI project:***

1. In the View List under **Organization**, click **Packages**.
2. Right-click the **Packages** explorer and then click **New InstallShield Project Package (.ism)**. The **Select the installation file for this package** dialog box opens.
3. Select the .ism file that you want to add to your Suite/Advanced UI project, and then click **Open**.

InstallShield adds the project to the Packages explorer. Configure its settings in the right pane. The Product Configuration setting (for a Basic MSI project package) and the Release setting (for a Basic MSI or InstallScript project package) enables you to select which release you want InstallShield to include in the Suite/Advanced UI installation.

To learn more about the inclusion of InstallScript packages in Advanced UI and Suite/Advanced UI projects, see Special Considerations for InstallScript Packages in Advanced UI and Suite/Advanced UI Projects.

Adding an Executable Package (.exe) to a Suite/Advanced UI Project

Project *• This information applies to Suite/Advanced UI projects.*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add executable packages (.exe) to a Suite/Advanced UI project. This format of Suite/Advanced UI package is recommended only if one of the preferred format types is not available. For more information, see Guidelines for Adding Packages to an Advanced UI or Suite/Advanced UI Project.

Task

To add an .exe package to your Suite/Advanced UI project:

1. In the View List under **Organization**, click **Packages**.
2. Right-click the **Packages** explorer and then click **New Executable Package (.exe)**. The **Select the installation file for this package** dialog box opens.
3. Select the .exe file that you want to add to your Suite/Advanced UI project, and then click **Open**. The **Add Files for This Package** dialog box opens.
4. Specify whether you want to include additional folders and files with your package file, and if so, whether you want to include dynamically linked files.

InstallShield adds the package, plus any additional folders and files, in the Packages explorer.

Special Considerations for Configuring Executable Packages in a Suite/Advanced UI Project

The following tasks are recommended when you are configuring an executable package in a Suite/Advanced UI project:

- Define appropriate conditions that direct the Suite/Advanced UI installation how to behave under various scenarios—for example, evaluating whether the package is already installed and checking for requirements and dependencies. For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.
- Suppress the user interface of the executable package and have the Suite/Advanced UI installation launch it silently. Otherwise, at run time, the installation shows the Suite/Advanced UI user interface and the separate user interface of the .exe package, presenting two disparate user interfaces. For more information, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Adding a Web Deploy Package to a Suite/Advanced UI Project

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add Web Deploy packages (.zip) to a Suite/Advanced UI project. At run time, the Suite/Advanced UI installation deploys the Web Deploy packages to IIS Web servers or the cloud based on input from end users. The Web Deploy packages can be created through IIS or a Web application development environment such as Visual Studio.

Task ***To add a Web Deploy package to your Suite/Advanced UI project:***

1. In the View List under **Organization**, click **Packages**.
2. Right-click the **Packages** explorer and then click **New Web Deploy Package (.zip)**. The **Select the installation file for this package** dialog box opens.
3. Select the Web Deploy package file that you want to add to your Suite/Advanced UI project, and then click **Open**. The **Add Files for This Package** dialog box opens.
4. Specify whether you want to include additional folders and files with your package file, and if so, whether you want to include dynamically linked files. Typically a Web Deploy package is self-contained, and it requires only the .zip file.

InstallShield adds the package, plus any additional folders and files, in the Packages explorer.

Special Considerations for Configuring Web Deploy Packages in a Suite/Advanced UI Project

Life Cycle of Web Deploy Packages

Web Deploy packages are typically intended to behave similarly to first-time installations or to install over existing earlier or identical versions. As a side effect, Web Deploy packages do not handle uninstallation. Thus, Suite/Advanced UI installations do not offer maintenance for this type of package. It is recommended that you avoid creating a Suite/Advanced UI installation that combines one or more Web Deploy packages with one or more traditional packages (that do offer maintenance) as primary packages.

Suite/Advanced UI Properties vs. Hard-Coded Values for Web Deploy Configuration Settings

When you select a Web Deploy package in the Packages view, InstallShield displays package-related settings in the right pane. The Configuration area on the Common tab of the right pane includes several configuration-related settings for Web Deploy, such as the destination—which is often a remote server—and credentials. In addition, the

Configuration area also includes settings for the parameters that are defined in the Web Deploy package's parameters XML file. By default, InstallShield uses Suite/Advanced UI properties for all of the configuration settings so that end users can set them at run time through the wizard interface or the command line.

If you want to include a default value for a property, use the Property Manager view to specify that value for its corresponding property.

In some cases, you may want to use hard-coded values for one or more parameters. For example, if a particular Web Deploy parameter needs to be set to a specific value, you may want to replace the parameter's default property name with a hard-coded value.

About the Wizard Interface for Configuration of Web Deploy Packages

When you add a Web Deploy package to a Suite/Advanced UI project, InstallShield automatically adds a predefined Web Deploy wizard page to your project for that package. The Web Deploy wizard page enables the end user to specify whether the package will be deployed to a local IIS server, a remote server, or a cloud-based server. It also uses a built-in Web Deploy–related action that enables the end user to load the configuration settings from a publisher profile file (.publishsettings).

If you want end users to be able to configure the parameters that are defined in your Web Deploy package's parameters XML file, you can add controls to the wizard interface as needed, and associate those controls with the properties that are specified for the corresponding parameter settings in the Packages view.

Adding a Sideloading UWP App Package (.appx) to a Suite/Advanced UI Project



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*



Note • *The ability to create and build a Suite/Advanced UI installation that includes a sideloading UWP app package requires Windows Vista or later or Windows Server 2008 or later on the machine that has InstallShield or the Standalone Build.*

InstallShield lets you add sideloading UWP app packages (.appx) to a Suite/Advanced UI project.

Sideloading an app is the process of installing an app without obtaining it through the Windows Store. This type of app is sometimes distributed to enterprise environments. Following are requirements for sideloading apps:

- The target system must have Windows 8 or later or Windows Server 2012 or later.
- The **Allow all trusted applications to install** group policy setting must be enabled on the target system.

For additional requirements for sideloading and running these types of apps, see the Windows documentation on the MSDN Web site.

Note that InstallShield does not have support for including UWP app packages that target the ARM processor architecture in a Suite/Advanced project.



Task ***To add a UWP app package (.appx) to your Suite/Advanced UI project:***

1. In the View List under **Organization**, click **Packages**.
2. Right-click the **Packages** explorer and then click **New Sideloading UWP App Package (.appx)**. The **Select the installation file for this package** dialog box opens.
3. Select the UWP App package that you want to add to your Suite/Advanced UI project, and then click **Open**. The **Add Files for This Package** dialog box opens.
4. Specify whether you want to include additional folders and files with your package file, and if so, whether you want to include dynamically linked files. Typically a UWP app package is self-contained, and it requires only the UWP App package and possibly the certificate file (.cer).

InstallShield adds the package, plus any additional folders and files, in the Packages explorer.

Note that when you add a sideloading app package to a Suite/Advanced UI project, InstallShield automatically adds an eligibility condition for that package to check for the presence of Windows 8 or later or Windows Server 2012 or later. This eligibility condition ensures that the Suite/Advanced UI installation does not attempt to add the package to systems that have earlier versions of Windows.

Static vs. Dynamic Files for Packages in an Advanced UI or Suite/Advanced UI Project



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are adding or configuring an .msi, .msp, or .exe package in an Advanced UI or Suite/Advanced UI project, you can indicate whether the package requires additional files that are located near the package file. For example, if a package that you are adding is an uncompressed .msi package, you may need to include other files—such as .cab files and uncompressed data files in nearby subfolders—along with the package file.

InstallShield supports several ways of including the additional files for a package in your project:

- **Use static links for all of the additional files**—This method is useful if the package has been finalized, and no other changes will be made. This method may also be useful if the names of all of the additional files are not likely to change while you are designing the Advanced UI or Suite/Advanced UI installation. At build time, InstallShield copies all of the package's static files that are listed in the Packages view into your Advanced UI or Suite/Advanced UI installation.

- **Use dynamic links for all of the additional files**—With this method, you specify the source location for the additional files. At build time, InstallShield copies all of the files that are present in the source location into your Advanced UI or Suite/Advanced UI installation.

  This method is useful if you are authoring both the package and the Advanced UI or Suite/Advanced UI installation, and the list of additional files that the package requires is likely to change between builds.

- **Use a combination of static and dynamic file links for the additional files**—With this method, you can use static links for all critical additional files whose names are known at design time, and use dynamic links for any other additional files. At build time, InstallShield copies all of the package's static files that are listed in the Packages view into your Advanced UI or Suite/Advanced UI installation. In addition, InstallShield also copies all of the dynamic files as needed.



Note *• InstallShield always includes a static link for the package file (.msi, .msp, InstallScript package, .exe, or other package file). Therefore, the name of the package file should be consistent from one build to another.*



Important *• Dynamic file linking should be used with caution. If you inadvertently delete a dynamically linked file from the source folder that your dynamic link references, that file is not included in your release the next time you build it, and InstallShield does not display any build warning or error. Your product may install without any issues, but it may not work as expected, since the dynamically linked file that was inadvertently deleted is no longer being installed.*

Distinguishing Static Source Locations from Dynamic Source Locations in InstallShield

When a statically linked folder is displayed within the Packages view, the folder's icon is a plain yellow folder.

When a dynamically linked folder is displayed within the Packages view, the folder's icon includes an image that indicates that it is dynamically linked:



InstallShield adds an arrow to the folder image—in addition to the dynamic file image—on the icon of a folder that contains one or more dynamically linked subfolders.



You can click either of those special folder icons to configure settings such as the source location for the content. You can also set up filters that you want InstallShield to use at build time to include or exclude certain files in the build.

Creating a Dynamic Link in an Advanced UI or Suite/Advanced UI Project



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

You can use the Packages view to create a dynamic link for the additional files that a package in the project requires.

You can include dynamic file links when you add your package to the project. You can also add them to your project later. The following sections provide instructions for both procedures.

Creating a Dynamic File Link When Adding a Package

When you add a package to your project in the Packages view, InstallShield displays the Add Files for This Package dialog box. This dialog box lets you specify whether the package file that you are adding requires other files and folders that are near the package that you are adding. For example, if you are adding an uncompressed .msi package to your project, you may need to include other files as well as folders that contain files that the .msi package installs on target systems. These folders and other files are typically in the same folder that contains the .msi package.

Do one of the following when InstallShield displays the Add Files for This Package dialog box:

- If you want to include only static files, clear the **Add files dynamically found during build** check box on this dialog box.
- If you want to include any dynamic files, with or without static files, select the **Add files dynamically found during build** check box on this dialog box.

Then select the option that describes where the source files are in relation to the package file. For more information about the various options, see Add Files for This Package Dialog Box.

Creating a Dynamic File Link After a Package Has Been Added



Task ***To add a dynamic file link for additional files after the package has been added to the project:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, find the package that requires the additional files, and click its corresponding folder that should contain the dynamic link. The **Files** pane shows the list of current files and dynamic folders.
3. Right-click in the **Files** pane and then click **Add Dynamic Link**. The **Browse for Folder** dialog box opens.
4. Browse to the folder that contains the files that you want to include dynamically, and then click **OK**.

Configuring a Dynamic File Link

Once you have added a dynamic file link to your project, you can configure its settings, which includes filter criteria for including files in and excluding files from the dynamic link. To learn more, see Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project

Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Once you have added a dynamic link to your project, you can specify filter criteria for including files in and excluding files from the dynamic link.

Task ***To specifying filter criteria for a dynamic file link:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, select the dynamically linked folder.
3. In the **Files** pane, click the folder for which you want to configure filter criteria.
4. In the **Dynamic Link** pane, review the subsettings under the **Filter** setting.
5. Do one of the following:
   - To add a filter:
     a. In the **Filter** setting, click the **New filter** button, and then select **Include File**, **Exclude File**, **Include Folder**, or **Exclude Folder**, depending on the type of filter you want to add. InstallShield adds a new row that you can use to define the filter.
     b. In the new row, specify the name or the pattern for the file or folder.

        To indicate a wild-card character, use an asterisk (*). For example, to include all .htm files, you could enter ***.htm** in an **Include File** setting. To include all files, you could enter ***.***.

        To refer to the root folder (the one that contains the package file) in a path, use a dot and a backslash:

        .

        For example, to refer to a **ReadMe.txt** file in a subfolder called **MyDirectory**, where **MyDirectory** is a subfolder of the folder that contains the package, you could use the following filter:

        .\MyDirectory\ReadMe.txt
   - To modify a filter: In the setting for the filter that you want to modify, enter the appropriate name or pattern for your filter.
   - To change the order of the filter criteria: In the setting for the filter that you want to move, click the **Move Filter** button, and then click **Move Up** or **Move Down**.

- To remove a filter: In the setting for the filter that you want to remove, click **Delete this Filter** button.

How an Advanced UI or Suite/Advanced UI Installation Evaluates the Filter Criteria at Build Time

At build time, InstallShield uses the filters that are set up for a dynamically linked folder to determine which folders and files to include in the release for a particular package, and which ones to exclude.

When InstallShield checks the dynamic link's source folder (the folder that contains the package), it reviews the filters in the order that they are listed under the Filter setting, from top to bottom. The first file filter that matches a file is the one that determines whether the file is included in the build. Similarly, the first folder filter that matches a folder is the one that determines whether the folder's files are included in the build.

For example, if the first filter in the list is as follows, InstallShield includes all files that are in the source folder, including a file called **ReadMe.txt**:

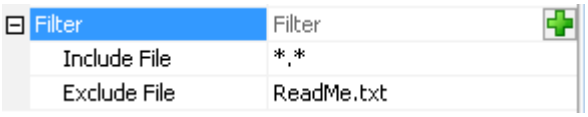


Figure 5-2: The *.* Include File filter overrides the ReadMe.txt Exclude File filter since the *.* filter is listed above the ReadMe.Txt filter.

If you move the **ReadMe.txt** filter exclusion above the *.* inclusion filter, the build includes all files except any that are named **ReadMe.txt**.

If a dynamically linked folder does not contain any filters, InstallShield includes all of the files that are in the source folder but none of the files in its subfolders.

Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

An InstallShield prerequisite is an installation for a product or technology framework that is required by your product. You can add any of the existing InstallShield prerequisites (.prq files) to your Advanced UI and Suite/Advanced UI projects. You can also create your own InstallShield prerequisites, and add them to your Advanced UI and Suite/Advanced UI projects.



Task ***To include an InstallShield prerequisite (.prq file) in an Advanced UI or Suite/Advanced UI project:***

1. In the View List under **Organization**, click **Packages**.
2. Right-click the **Packages** explorer, and then click **Import Prerequisite (.prq)**.

InstallShield adds the InstallShield prerequisite to the Packages explorer as an .msi package, an .msp package, or an .exe package, depending on the type of file that is configured to run for the prerequisite.

If the InstallShield prerequisite has dependencies (that is, if one or more other .prq files are specified as dependencies in the InstallShield prerequisite that you are adding to the Advanced UI or Suite/Advanced UI project), InstallShield automatically adds the dependency prerequisites as separate packages in the Packages explorer.



Tip *• When you add InstallShield prerequisites to an Advanced UI or Suite/Advanced UI project, InstallShield automatically configures default values for the prerequisite package's settings. You can change these settings as needed, just as you can change the settings for packages in your Advanced UI or Suite/Advanced UI project. For more information, see Configuring Settings for a Package in an Advanced UI or Suite/Advanced UI Project.*

Reviewing the Detection and Evaluation Conditions of an InstallShield Prerequisite That Is Included as a Package in an Advanced UI or Suite/Advanced UI Project

Once you have imported an InstallShield prerequisite into an Advanced UI or Suite/Advanced UI project, review the detection and evaluation conditions of the package, and make changes as needed.

For example, if you import an InstallShield prerequisite that targets version 5.0 and all later versions of the Windows operating system, you may want to change the conditions that InstallShield configures by default when you include the prerequisite in your project; you could replace the multiple platform conditions with a single platform condition in which the OS Version condition subsetting is **5.0-** (to indicate version 5.0 and later).

Primary Packages vs. Dependency Packages in Advanced UI and Suite/Advanced UI Projects



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where appropriate.



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Each package in Advanced UI and Suite/Advanced UI projects needs to be identified as either a primary package or a dependency package. The package type identifies whether the presence of that package on a target system should influence whether the Advanced UI or Suite/Advanced UI installation runs in first-time installation mode or maintenance mode:

- **Primary package**—A primary package is a main part of the Advanced UI or Suite/Advanced UI installation.

  At run time, if all of the primary packages in the installation are absent from the target system, the installation runs as a first-time installation. If any of the primary packages are present on the target system, the installation runs in maintenance mode.

- **Dependency package**—The presence or absence of a dependency package on the target system does not influence which mode is used to run the installation.

The Package Type setting that is available when you select a package in the Packages view indicates whether the package is a primary package or a dependency package.

In Suite/Advanced UI projects, you can change the value of the Package Type setting as needed for any packages in your project. However, in Advanced UI projects, this setting is read-only, since an Advanced UI project has support for only one primary package.

Note that when you import an InstallShield prerequisite into an Advanced UI project or a Suite/Advanced UI project, InstallShield adds the InstallShield prerequisite to the project as a dependency package. In a Suite/Advanced UI project, you can use the Package Type setting to override this default value, and specify that the package is a primary package if appropriate. However, an InstallShield prerequisite in an Advanced UI project is always identified as a dependency package, and this default behavior cannot be overridden in the Professional edition of InstallShield.

Specifying the Installation Order of Packages and Transactions in an Advanced UI or Suite/Advanced UI Project



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where appropriate.



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

If your Suite/Advanced UI project includes more than one package, transaction, or InstallShield prerequisite, the Packages view is where you specify the order in which the packages should be installed and the transactions should be launched at run time on target systems.

Similarly, if your Advanced UI project includes a package and one or more InstallShield prerequisites, the Packages view is where you specify the order in which the packages should be installed at run time on target systems.

This Packages view lists the packages, InstallShield prerequisite packages, and (if supported) transactions in the same order in which the Advanced UI or Suite/Advanced UI installation launches them at run time.

Task ***To modify the order in which the packages should be installed on target systems:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, right-click the package or transaction that you want to move and click **Move Up** or **Move Down**.

Configuring Settings for a Package in an Advanced UI or Suite/Advanced UI Project

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you add a package to an Advanced UI or Suite/Advanced UI project, you can configure its settings to specify details such as the display name and the conditions under which the Advanced UI or Suite/Advanced UI installation should launch the package.

Task ***To configure a package in your Advanced UI or Suite/Advanced UI project:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, click the package that you want to configure.
3. Use the settings on the **Common** tab to define conditions for the package and configure other settings.

For information on any of the package settings, see Common Tab.

At run time, the **Setup.exe** file launches each package based on the conditions that you have defined and the order in which you listed the packages in the Packages view.

Tip • *You can use an asterisk as a placeholder for the current package's product code package code, and product version when you are defining eligibility and detection MSI package conditions, as long as the package that you are configuring is an .msi package.*

You can also use an asterisk as a placeholder for the current package's upgrade code and either the minimum or maximum version number when you are defining eligibility and detection MSI upgrade conditions, as long as the package that you are configuring is an .msi package.

Associating a Package in an Advanced UI or Suite/Advanced UI Project with a Feature



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Each package that is included in an Advanced UI or Suite/Advanced UI project should be associated with a feature in the project.



Task ***To associate a package in your Advanced UI or Suite/Advanced UI project that requires the target system to be restarted:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, select the package that you want to configure.
3. Click the **Features** tab.
4. Select the check box of each feature that you want to contain the package.

Sharing Common Packages Among Different Advanced UI and Suite/Advanced UI Installations



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to share common packages among multiple Advanced UI and Suite/Advanced UI installations. If two or more Advanced UI and Suite/Advanced UI installations share a package, the package remains on the target system until all of the Advanced UI and Suite/Advanced UI products are removed.

Guidelines for Determining Whether to Mark a Package as Shared

When you are configuring a package for an Advanced UI or Suite/Advanced UI installation, you can mark it as shared. If you mark a package as shared, ensure that you also use the same package GUID in all Advanced UI and Suite/Advanced UI projects that are sharing the package.

The shared package capability is typically reserved for dependency packages, rather than for primary packages. Note that widely distributed packages—such as a package that installs the .NET Framework—are usually not marked as shared, since these are most often left installed on target systems.

Note that shared capability is available for most types of packages (.msi, .msp, .exe, .appx, InstallScript, Basic MSI project, and InstallScript project). However, it is not available for Web Deploy packages, which are not typically intended to be uninstalled.

Specifying Whether a Package Is Shared



Task ***To specify whether a package in an Advanced UI or Suite/Advanced UI project is shared:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, select the package that you want to configure.
3. In the **Shared** setting, select the appropriate value:
   - **No**—The package will not be shared among other Advanced UI or Suite/Advanced UI projects. This is the default value.
   - **Yes**—The package will be shared among other Advanced UI or Suite/Advanced UI projects.

     If you select **Yes** for the **Shared** setting, ensure that you also use the same package GUID in all Advanced UI and Suite/Advanced UI projects that are sharing the package; the **Package GUID** setting is where you can view and modify the GUID of a package.

Run-Time Behavior for Shared Packages

At run time, when an Advanced UI or Suite/Advanced UI installation installs a shared package that is not already present, the installation creates a new registry key for the package GUID of the shared package under the following registry key:

- For 32-bit systems: HKEY_LOCAL_MACHINE\SOFTWARE\InstallShield\SuiteInstallers\Parcels
- For 64-bit systems: HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\InstallShield\SuiteInstallers\Parcels

The installation also adds its Suite GUID as registry data in the package GUID registry key. When other Advanced UI or Suite/Advanced UI installations need to install the same shared package, the Suite GUIDs of those Advanced UI or Suite/Advanced UI installations are added to registry data in the package GUID registry key. During uninstallation of any of the Advanced UI or Suite/Advanced UI products, the Suite GUIDs are removed from the package GUID registry key. The shared package and the package GUID registry key remain on the target system until the last Advanced UI or Suite/Advanced UI product is removed.

If two Advanced UI or Suite/Advanced UI installations share different versions of the same package, the later version of the package is the one that is installed on target systems after both Advanced UI or Suite/Advanced UI installations are run, regardless of which one was run first. In addition, the installations update the Parcels registry key as needed; the shared package and the package GUID registry key remain on the target system until the last Advanced UI or Suite/Advanced UI product is removed.

Specifying a Run-Time Location for a Specific Package in an Advanced UI or Suite/Advanced UI Project



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to specify a different run-time location for each package in your project. For example, you can have a package and its files stored on the source media, or downloadable if required from the Web. Or, you can have a package compressed into the Advanced UI or Suite/Advanced UI **Setup.exe** file and extracted at run time if necessary.



Task ***To specify a different run-time location for each package in your project:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, select the package that you want to configure.
3. In the **Location** list, select the appropriate option.

Note that the location that you specify can be overridden at the release level. To avoid overriding the value that you selected for an individual package, the Package Location setting at the release level must be set to Follow Individual Selections. For more information, see Specifying the Run-Time Location for Advanced UI or Suite/Advanced UI Packages at the Release Level.



Tip *• If you are configuring an Advanced UI or Suite/Advanced UI project for an update setup launcher, the run-time location of the packages must be either extracted from the setup launcher or downloaded from the Web. The update setup launcher cannot rely on packages that are stored on the source media.*

For more information, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.

Configuring Package Operations for an Advanced UI or Suite/Advanced UI Installation



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Advanced UI projects include support for .exe packages only if they are included in the project from an InstallShield prerequisite.



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

An Advanced UI or Suite/Advanced UI operation is the state in which a package in the Advanced UI or Suite/Advanced UI installation is running. The different types of operations are install, remove, repair, modify, and maintenance.

When you add a package to an Advanced UI or Suite/Advanced UI project, you can configure various operation settings for the package. The settings let you specify whether each of the operations are applicable to the package. The settings also let you enter command lines that you want the Advanced UI or Suite/Advanced UI installation to use when launching the package in each mode. You can specify separate command lines: one for when the Advanced UI or Suite/Advanced UI installation is running with a user interface, and one when it is running silently (without a user interface).

Install Operation

An Advanced UI or Suite/Advanced UI installation runs an install operation to install the package onto a target system. This occurs during either a first-time installation, or when a feature is selected to be installed during maintenance. All packages typically have an install operation.

When an Advanced UI or Suite/Advanced UI installation runs an install operation for an .msi, .msp, or InstallScript package, the Advanced UI or Suite/Advanced UI installation automatically launches the package silently. However, when an Advanced UI or Suite/Advanced UI installation runs an install operation for an .exe package, the Advanced UI or Suite/Advanced UI installation cannot automatically launch the package silently, since different types of .exe packages may use different ways of running without a user interface. If you want the Advanced UI or Suite/Advanced UI installation to run an .exe package silently, ensure that you enter the appropriate command line for the package's install operation subsettings.

If any of your Advanced UI or Suite/Advanced UI packages should be configurable by end users, create wizard pages that let end users configure properties, and pass the associated properties to the packages through the command lines.

Remove Operation

An Advanced UI or Suite/Advanced UI installation runs a remove operation to uninstall the package from a target system. Packages that are marked as a primary package typically have a remove operation; packages that are marked as a dependency package often do not have a remove operation, since they are typically left on target

systems. A package that installs the .NET Framework is an example of a package that would be marked as a dependency package. The Package Type setting for the package identifies whether the package is a primary package or a dependency package.

An Advanced UI or Suite/Advanced UI installation runs a remove operation when an Advanced UI or Suite/Advanced UI feature is turned off during maintenance, or when the Advanced UI or Suite/Advanced UI product is uninstalled. Like during installation, an .exe package's command lines should run the package silently.

Note that the remove operation does not apply to .msp packages; removing an .msp package requires removing the base .msi package.

Repair Operation

An Advanced UI or Suite/Advanced UI installation runs a repair operation to invoke the repair functionality that some packages support. If you are configuring an Advanced UI or Suite/Advanced UI package that supports repair, configure the package's repair operation setting to allow repairs. The Advanced UI or Suite/Advanced UI installation runs the repair operation only when the Advanced UI or Suite/Advanced UI installation is being repaired as part of maintenance, and only if the package is already installed.

If you are configuring a repair operation for an .msi package, you do not need specify any special command lines to trigger a silent repair, since the Advanced UI or Suite/Advanced UI installation can trigger the repair automatically if appropriate. However, if you are configuring a repair operation for an .exe package, you must enter the appropriate command line to trigger the repair. For example, for a Basic MSI **Setup.exe** package, you could enter the following command line in the EXE Command Line and the EXE Silent Command Line subsettings under the Repair operation setting:

```
[SystemFolder]msiexec.exe /f {8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC} /qn
```

The GUID in braces is the product code of the Windows Installer package.

Note that the repair operation does not apply to .msp packages.

Modify Operation

An Advanced UI or Suite/Advanced UI installation runs a modify operation to perform feature selection. If you are configuring an Advanced UI or Suite/Advanced UI package that offers feature selection, configure the package's modify operation setting to allow modifications. The Advanced UI or Suite/Advanced UI installation runs the modify operation when Advanced UI or Suite/Advanced UI feature selection changes indicate that a package's features should be added or removed.

Note that the modify operation does not apply to .msp packages.

Maintenance Operation

An Advanced UI or Suite/Advanced UI installation runs a maintenance operation in scenarios such as the following ones:

- An end user chooses to modify the product through its entry in Add or Remove Programs.
- An end user reruns the Advanced UI or Suite/Advanced UI installation.

The maintenance operation differs from the other types of operations because it is not controlled through an operation setting.

Using Custom Folder Names for Packages in Advanced UI and Suite/Advanced UI Installations

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you build an Advanced UI or Suite/Advanced UI installation, InstallShield creates a folder for each package that is included in the installation; these folders are created in the same folder that contains the Advanced UI or Suite/Advanced UI setup launcher. By default, the name of each folder is a GUID that InstallShield generates.

The Packages view in InstallShield now lets you override the GUID name with a user-friendly name for each package folder.

Task

To specify your own text for the name of a package folder:

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, expand the node for the package whose folder name you want to customize.
3. Right-click the **Package Files** folder, and then click **Rename**.

   Note *• If you leave the default name (**Package Files**), InstallShield uses a GUID for the name of the package folder.*
4. Enter a new name for the folder. If you customize more than one package folder name, ensure that each folder name is different.

At build time, InstallShield uses the text that you entered instead of a GUID for the name of the package folder when creating the release.

Customizing the Behavior of a Suite/Advanced UI Installation

Project *• This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

An important aspect of creating a Suite/Advanced UI installation is customizing it for your product's requirements and your end users' needs. InstallShield lets you extend the functionality of your Suite/Advanced UI installation to perform various run-time tasks that are outside the scope of the packages that you are including in the installation. For more information, see:

- Enabling Windows Roles and Features During a Suite/Advanced UI Installation
- Using Actions to Extend the Behavior of a Suite/Advanced UI Installation

Enabling Windows Roles and Features During a Suite/Advanced UI Installation

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Some Windows functionality is disabled by default on target systems. In addition, some end users, or perhaps IT administrators, may choose to disable some Windows functionality. To manually enable or disable such Windows roles and features, they can use the **Turn Windows features on or off** link in the Programs and Features Control Panel.

If a particular package in your Suite/Advanced UI installation requires that one or more Windows roles or features be enabled on target systems, you can specify the required roles and features when you are configuring the package in your Suite/Advanced UI project. At run time, if an eligible package that is being installed requires one or more Windows roles or features that are disabled, the Suite/Advanced UI installation enables them before launching the package.

For example, your Suite/Advanced UI project may have a package that installs an IIS Web site that requires that the Internet Information Services feature in Windows be turned on. Another package in the same project may require that the PowerShell feature be turned on. When you are configuring the settings of those packages in your project, you can specify the Windows features that are required; at run time, if a package is eligible to be installed on a target system but any of its required Windows features are disabled, the Suite/Advanced UI installation enables those disabled Windows features before launching it. If the package is not eligible, its required Windows features are not enabled.

Note that during uninstallation, any Windows roles and features that the Suite/Advanced UI installation had enabled during installation are not removed; this helps avoid breakage of other products on the target system that require those roles and features.

InstallShield has built-in support for enabling several Windows features:

- Internet Information Services
- PowerShell
- .NET Framework 3.x

InstallShield also lets you specify additional Windows roles and features that a package requires.



Task ***To specify a Windows role or feature that you want to be enabled if a particular package is eligible for installation:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, click the package that you want to configure.
3. On the **Common** tab, in the **Windows Features** setting, click the **Add New Windows Feature** button, and then click one of the available feature options:
   - **Internet Information Services**
   - **PowerShell**
   - **Microsoft .NET Framework 3.x**
   - **Custom**—This option lets you specify any Windows feature that is not listed as an option in this setting.

   InstallShield adds a new **Windows Feature** row under the **Windows Features** setting, and configures it as needed for any of the built-in feature options (Internet Information Services, PowerShell, or Microsoft .NET Framework 3.x). For the **Custom** option, the **Windows Feature** row contains a placeholder string for the name of a feature.
4. If you selected one of the built-in options, leave the value that InstallShield adds to the **Windows Feature** setting as is.

   If you selected the **Custom** option: In the **Windows Feature** setting, enter the name of the Windows role or feature that you want to be enabled. The name that you enter for any custom option should be the string that tools such as Deployment Image Servicing and Management (**DISM.exe**) and Package Manager (**Pkgmgr.exe**) use to identify the role or feature on a particular version of Windows.

   For a list of available Windows roles and features on each version of Windows for which you are supporting, see Microsoft TechNet.

Special Considerations for Enabling Windows Roles and Features

Availability of Windows Roles and Features on Various Versions of Windows

The support for enabling Windows roles and features is available for target systems that have Windows Vista or later or Windows Server 2008 or later. The list of Windows roles and features that can be enabled on a given target system varies, depending on the version of Windows that is present. In addition, different versions of Windows may use different strings for the same role or feature.

If a Suite/Advanced UI installation attempts to enable a Windows role or feature that is not available on a target system, the Suite/Advanced UI installation fails and writes error 0x800f080c in the log file.

Thus, in some scenarios in which you are specifying required Windows roles and features, you may want to use a package's Eligibility Condition setting to identify the appropriate operating system versions that correspond with the required Windows role or feature that you are selecting for the package. To target multiple versions of Windows, each with a different string for the name of the Windows role or feature that you are enabling, consider adding the package to your project multiple times. For each instance of the package, use the package's Windows Features and Eligibility Condition settings to specify the appropriate string for the corresponding target operating systems.

Note that some roles and features may not only be disabled on some versions of Windows; they also may not be installed. If a Suite/Advanced UI installation is configured to enable a Windows role or feature that is not installed on a target system, the Suite/Advanced UI installation attempts to install the role or feature. If the role or feature cannot be installed, the Suite/Advanced UI installation fails. An example of a Windows feature to which this scenario may apply is the .NET Framework 3.x feature, which may not be installed by default on Windows 8 systems. The payload for this feature is available only through Windows Update. If an Internet connection is not available, or if the system is configured to obtain updates through Windows Server Update Services instead of Windows Update, the Suite/Advanced UI installation cannot install and enable the .NET Framework 3.x feature.

Identifying Dependencies for a Required Windows Role or Feature

Some Windows roles and features depend on other Windows roles or features. One way to determine whether a disabled role or feature has any dependencies is to run the following command from an elevated Command Prompt window (which you can launch by right-clicking a shortcut for **cmd.exe** and then clicking **Run as admin**):

```
dism /online /enable-feature FEAT-NAME
```

In this example, FEAT_NAME is the name of the disabled feature that you are checking. If the aforementioned command works, the feature has no dependencies. If it fails, the error lists the dependencies.

If the role or feature that you are enabling for a package requires one or more roles or features that may be disabled on target systems, ensure that you also specify the dependency features in the package's Windows Features setting.

Run-Time Behavior for Enabling a Windows Role or Feature

If a Suite/Advanced UI installation enables a Windows role or feature on some target systems, the InstallationProgress wizard page of the Suite/Advanced UI installation can show the progress of the role or feature being enabled. This progress information is available on target systems that have Windows 8 or later or Windows Server 2012 or later. On these target systems, a Suite/Advanced UI installation uses DISM APIs to enable roles and features, and these APIs have support for reporting the progress information.

On Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 target systems, a Suite/Advanced UI installation uses **Pkgmgr.exe** to enable roles and features. **Pkgmgr.exe** does not have support for reporting progress information; therefore, on these systems, the progress bar cannot updated as roles and features are being enabled.

Although **DISM.exe** is available on Windows 7 and Windows Server 2008 R2 systems, the API support is not shipped on these versions of Windows. In addition, the API DLL dependencies are not available for redistributing to target systems.

Using Actions to Extend the Behavior of a Suite/Advanced UI Installation



Project *• This information applies to Suite/Advanced UI projects.*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

You may require your Suite/Advanced UI installation to perform various run-time tasks that are outside the scope of the packages that you are including in the installation. For example, you may need the Suite/Advanced UI installation to do one or more of the following:

- Install an application before the user interface of the Suite/Advanced UI installation is displayed.

  One sample scenario in which this may be necessary is when one of the packages in your Suite/Advanced UI installation runs SQL scripts to install an Oracle database. Before this package is run, you may want the Suite/Advanced UI interface to display a wizard page that lets end users select the appropriate server from a list of servers that are available on the network. This type of UI support requires that ODBC drivers be installed on the target system.

- Search the target system for the presence or absence of a particular product, technology, folder, file, registry entry, or other item.

  Support for searching the target system enables you to trigger behavior based on whether certain conditions are met. For example, if a particular file is missing on a target system, you can set a Suite/Advanced UI property accordingly, and then use that property to set a property in one or more of the packages in the Suite/Advanced UI installation.

- Configure the target system before or after running a package in the Suite/Advanced UI installation.

  In some scenarios—for example, if your Suite/Advanced UI installation installs an IIS application on servers—you may want to enable IIS-related Windows roles and features on target systems. In other scenarios, you may want to edit configuration files, registry data, or other items on the target system based on which packages in the Suite/Advanced UI installation are selected to be run by end users.

- Perform some sort of initialization that end users trigger when using a particular wizard interface control, or when a wizard page or secondary window is opened or closed.

  For example, you may want to add an action to a combo box control on a wizard interface to dynamically populate the values in the combo box.

To extend the capabilities of a Suite/Advanced UI installation and make it possible to perform those sorts of tasks and more, you can use the Events view in your Suite/Advanced UI project to create actions that run executable files, call DLL functions, run PowerShell scripts, set a Suite/Advanced UI property, run InstallScript code, or call a public method in a managed assembly at run time.

Task ***To author an action and add it to your Suite/Advanced UI project:***

1. Create your own .exe file (if appropriate), DLL file, PowerShell script, InstallScript file (.rul), or managed assembly (written in managed code such as Visual Basic .NET or C#) that performs the functionality that your installation requires.

   For an InstallScript action, you can use the InstallScript view in your project.

2. In the Events view, create an action that uses the file that you created for step 1. For more information, see Adding an Action to a Suite/Advanced UI Project.

3. Decide when your action should run, and schedule it: add it to an event, add it to a wizard interface element, or assign it to an event in a specific package in your project. To learn more, see Scheduling a Suite/Advanced UI Action.

Working with an .exe File for an Action in a Suite/Advanced UI Installation

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

You can use an .exe action in a Suite/Advanced UI installation to launch an executable file that is included in your installation (as a temporary support file or installed with the product) or that already exists on the target system.

Following are examples of scenarios in which you may want to add an .exe action to your Suite/Advanced UI installation:

- You can add an .exe action that launches a **Setup.exe** installation that you or a third party created.
- You can add an .exe action that sets permissions on existing directories.
- You can add an .exe action that opens a ReadMe file in its associated application—perhaps Notepad or Adobe Reader.
- You can add an .exe action that opens a data file in your product.

The first step in adding an .exe action to a Suite/Advanced UI project is to create or acquire the .exe file, if applicable, or to create the document or other file that the executable file will launch.

Note that .exe actions do not have access to the running installation session. Thus, you cannot pass Suite/Advanced UI properties to the .exe file (except as command-line arguments) or back from the .exe file (except through external storage, such as the registry or a file).

Task ***To add an action that runs an executable file in a Suite/Advanced UI installation:***

1. In the View List under **Behavior and Logic**, click **Events**.
2. Right-click the **Actions** explorer and then click **New EXE**. InstallShield adds an executable-file action to the **Actions** explorer.
3. Enter a new name, or right-click it later and click **Rename** to assign it a new name. Use a name that helps you distinguish this action from other actions in your project.
4. Configure the action's settings as needed.

Once you have added the action to your project, schedule it as needed. For more information, see Scheduling a Suite/Advanced UI Action.

Note *• If you are launching an interactive application from an .exe action, you may want to add a condition to the action to prevent it from launching when the Suite/Advanced UI installation is running silently (without a user interface). You can use the* ***ISSilentInstall*** *property in this type of condition; if the Advanced UI or Suite/Advanced UI installation is running silently (without a user interface), this property is set to True.*

As an alternative, you may want to launch interactive applications from the user interface of your installation. Thus, instead of using an .exe action to launch a PDF file, you may want to add the ability to open the PDF file from one of the wizard pages in your installation's user interface.

Working with a DLL File for an Action in a Suite/Advanced UI Installation

Project *• This information applies to Suite/Advanced UI projects.*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Creating a DLL File for a DLL Action

The first step in adding a DLL action to a Suite/Advanced UI project is to create the DLL. You can create the DLL with any recent version of Visual C++ or any tool or language that supports COM and DLL exports.

The Suite/Advanced UI engine requires that each action have one entry point. To specify the entry point, enter it in the Function Name setting for the action in the Events view. The function should be defined as follows:

```
HRESULT __stdcall MyFunction(IDispatch *pAction);
```

To be able to call this function from the Suite/Advanced UI installation, you must include a definition file (.def) when you build the DLL to export the function properly. The following is the contents of a sample .def file. The name after LIBRARY should be the name that you are using for the DLL.

```
LIBRARY "MyActionLibrary"
EXPORTS
```

```
MyAction
```

The Suite/Advanced UI installation calls the function entry point when an action for an event or a package in the installation is scheduled to run. Your entry-point function should return ERROR_SUCCESS to indicate that the action finished successfully. If the action fails, it should return a non-zero value. The values are the same ones that Windows Installer custom actions return.

Note that although the Suite/Advanced UI installation allows any action to be performed, actions are run with the privileges that they are configured to have. Some actions may require administrator privileges (such as reading IIS 7.x configuration data). In such cases, in order for the action to access the required data successfully, the action would need to be configured to require administrator privileges; that is, Yes would need to be selected for the Requires Administrative Privileges setting for the action in the Events view. As an alternative, the Suite/Advanced UI installation's **Setup.exe** file could be run with administrator privileges, or it would need to include an administrator manifest. However, privileges should be elevated for as short of a duration as possible.

The IDispatch interface that is passed to the action's entry point implements the ISuiteExtension2 interface (which includes all of the ISuiteExtension methods and properties). To obtain a pointer to ISuiteExtension2, call the QueryInterface method on the IDispatch interface. The ISuiteExtension2 interface allows the action function to access the attribute parameter that is defined for the action in the Suite/Advanced UI project. Note that each action in the project receives a different interface pointer that is specific to each action instance. Therefore, the interface pointer that is passed to the entry point function should always be used and should not be saved across calls to the action DLL.

The interface is defined as follows:

```
interface ISuiteExtension2 : IDispatch {
  [propget, id(1), helpstring("Attribute")]
    HRESULT Attribute([in]BSTR bstrName, [out, retval]BSTR *bstrValue);

  [id(2), helpstring("method LogInfo")]
    HRESULT LogInfo([in]BSTR bstr);

  [propget, id(3), helpstring("Property")]
    HRESULT Property([in]BSTR bstrName, [out, retval]BSTR *bstrValue);

  [propput, id(3), helpstring("Property")]
    HRESULT Property([in]BSTR bstrName, [in]BSTR bstrValue);

  [id(4), helpstring("FormatProperty")]
    HRESULT FormatProperty([in]BSTR bstrValue, [out, retval]BSTR *bstrReturnValue);

  [id(5), helpstring("ResolveString")]
    HRESULT ResolveString([in]BSTR bstrStringId, [out, retval]BSTR *bstrReturnValue);

  [id(6), helpstring("ProgressMessage")]
    HRESULT SendProgressMessage([in]BSTR bstrMsg, [in]INT iCurrent, [in]INT iMax,
                                [in]EnumProgressFlags eFlags);
};
```

Suite/Advanced UI projects use the function prototype to pass the ISuiteExtension2 COM interface pointer to an action, which enables you to access the following functions:

- **LogInfo**

  ```
  HRESULT LogInfo([in]BSTR bstr);
  ```

  This method writes any information that is needed to the Suite/Advanced UI debug log, making it available for debugging or other informational purposes. The bstr parameter contains the string to be written to the log.

- **get_Property**

  ```
  HRESULT get_Property([in]BSTR bstrName, [out, retval]BSTR *bstrValue);
  ```

  This method retrieves the value of any property that is currently defined in the running Suite/Advanced UI installation. Properties that are not defined return an empty value. The bstrName parameter specifies the name of the property whose value is being obtained. The value for the property is returned in the bstrValue parameter.

- **put_Property**

  ```
  HRESULT put_Property([in]BSTR bstrName, [in]BSTR bstrValue);
  ```

  This method sets the value of a new property or changes the value of an existing property in the currently running Suite/Advanced UI installation. Passing an empty value effectively deletes the property. The bstrName parameter specifies the name of the property to set. The bstrValue parameter specifies the value to set for the specified property.

- **FormatProperty**

  ```
  HRESULT FormatProperty([in]BSTR bstrValue, [out, retval]BSTR *bstrReturnValue);
  ```

  This method resolves embedded formatted expressions in the string that is provided in the bstrValue parameter. The formatted value is returned in the bstrReturnValue parameter. To learn about the syntax that is available for formatted expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

- **ResolveString**

  ```
  HRESULT ResolveString([in]BSTR bstrStringId, [out, retval]BSTR *bstrReturnValue);
  ```

  This method resolves a Suite/Advanced UI string identifier into the corresponding string value for the currently running Suite/Advanced UI installation in the currently selected UI language. The bstrStringId parameter specifies the string identifier to resolve, and the resolved string is returned in bstrReturnValue. If no such string identifier exists, the returned string is empty.

- **SendProgressMessage**

  ```
  HRESULT SendProgressMessage([in]BSTR bstrMsg, [in]INT iCurrent, [in]INT iMax,
                              [in]EnumProgressFlags eFlags);
  ```

  This method updates the status message and the progress bar on the InstallationProgress wizard page of the Suite/Advanced UI installation. To update the progress bar, specify epfProgressValid for the eFlags parameter; to update the status message, specify epfMessageValid. To update both the status message and the progress bar, use the bitwise OR operator (|) with both flags.

Note that the get_Attribute method is not available for DLL actions in a Suite/Advanced UI installation, and it should not be called.

To access the ISuiteExtension2 interface from your DLL, you can use `#import` to incorporate the type library information from the **SetupSuite.exe** file that is installed with InstallShield. The path is:

*`InstallShield Program Files Folder`***\Redist\Language Independent\i386\SetupSuite.exe"**

For example, to import the type library in a VC++ project (such as in **stdafx.h**), use the following statement:

```
#import "C:\Program Files\InstallShield\2016\Redist\Language Independent\i386\SetupSuite.exe"
no_namespace raw_interfaces_only named_guids
```

Note that if InstallShield is installed to a different location, adjust the path in the #import statement accordingly.

Adding a DLL Action

Task ***To add an action that calls a function in a DLL file during a Suite/Advanced UI installation:***

1. In the View List under **Behavior and Logic**, click **Events**.
2. Right-click the **Actions** explorer and then click **New DLL**. InstallShield adds a DLL action to the **Actions** explorer.
3. Enter a new name, or right-click it later and click **Rename** to assign it a new name. Use a name that helps you distinguish this action from other actions in your project.
4. Configure the action's settings as needed.

Once you have added the action to your project, schedule it as needed. For more information, see Scheduling a Suite/Advanced UI Action.

Working with a PowerShell Script for an Action in a Suite/Advanced UI Installation

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Windows PowerShell is a .NET Framework–based command-line shell and script language that enables system administrators to automate system configuration tasks. InstallShield lets you include in Suite/Advanced UI installations actions that run PowerShell scripts (.ps1). You may want to add this type of action to a project to perform system configuration tasks at installation run time. Note that PowerShell actions require PowerShell 2.0 or later on target systems.

Adding a PowerShell Action

Task ***To add an action that runs a PowerShell script in a Suite/Advanced UI installation:***

1. In the View List under **Behavior and Logic**, click **Events**.
2. Right-click the **Actions** explorer and then click **New PowerShell**. InstallShield adds a PowerShell action to the **Actions** explorer.
3. Enter a new name, or right-click it later and click **Rename** to assign it a new name. Use a name that helps you distinguish this action from other actions in your project.
4. Configure the action's settings as needed.

If you select a PowerShell action in the Actions explorer or the Events explorer, InstallShield displays a PowerShell Script tab on the right; you can use this tab to edit your script.

Once you have added the action to your project, schedule it as needed. For more information, see Scheduling a Suite/Advanced UI Action.

Run-Time Requirements for a PowerShell Action

You can use the Suite/Advanced UI property **IS_CLR_VERSION** to identify a semicolon-delimited list of .NET Framework versions that the action should attempt to load to run your PowerShell script. To learn more about the **IS_CLR_VERSION** property, see Advanced UI and Suite/Advanced UI Property Reference.

Interacting with the Running Suite/Advanced UI Installation

Several cmdlets that let you interact with the running Suite/Advanced UI installation are available:

Table 5-2 • Cmdlets that Interact with a Running Suite/Advanced UI Installation

Cmdlet	Description
get-suiteproperty -name [Property Name]	This cmdlet gets the value of a Suite/Advanced UI property. The Name parameter is a positional parameter. The following sample code gets the value of the property called **MYPROPERTY**: `$Value = get-suiteproperty -name MYPROPERTY`

set-suiteproperty -name [Property Name] -value [value]	This cmdlet sets the value of a Suite/Advanced UI property.
	The following sample code uses the value of the **MYPROPERTY** property to set the value of a second property called **NEWPROPERTY**:
	`$Value = get-suiteproperty -name MYPROPERTY`
	`set-suiteproperty -name NEWPROPERTY -value $Value`

format-suiteproperties -Value [format string]	This cmdlet expands the value of a formatted expression. The Name and Value parameters are positional parameters. The following sample code resolves a string that contains a Suite/Advanced UI property called **NEWPROPERTY**. The property name and its surrounding square brackets are replaced by the value of the property at run time. `$myFormat = format-suiteproperties -Value "This is a text string with [NEWPROPERTY] embedded."`
resolve-suitestring -StringId [string ID]	This cmdlet resolves a Suite/Advanced UI string identifier with its corresponding string value in the currently selected UI language. If no such string identifier exists, the returned string is empty. The StringId parameter is a positional parameter. The following sample code resolves a string identifier called **IDS_MY_MESSAGE**: `$StringValue = resolve-suitestring -StringId IDS_MY_MESSAGE`

Table 5-2 • Cmdlets that Interact with a Running Suite/Advanced UI Installation (cont.)

Cmdlet	Description

trace-suiteinfo -LogMessage [log string]	This cmdlet writes any information that is needed to the Suite/Advanced UI debug log, making it available for debugging or other information purposes. The LogMessage parameter is a positional parameter.
	
	The following sample code resolves a string identifier called **IDS_MY_MESSAGE** and then writes its value to the Suite/Advanced UI debug log.
	
	`$StringValue = resolve-suitestring -StringId IDS_MY_MESSAGE`
	
	`trace-suiteinfo -LogMessage $StringValue`

To indicate that your script has succeeded, ensure that it returns 0. For example, you may want to end your script with the following:

```
exit(0)
```

Once you have added a PowerShell action in the Events view of your project, you can select it in the Actions explorer or the Events explorer, and use the PowerShell Script tab that InstallShield displays on the right to edit your script.

Working with an Action that Sets a Property in a Suite/Advanced UI Installation



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add to your Suite/Advanced UI project an action that sets a Suite/Advanced UI property at run time.



Task ***To add an action that sets a property in a Suite/Advanced UI installation:***

1. In the View List under **Behavior and Logic**, click **Events**.
2. Right-click the **Actions** explorer and then click **New Set Property**. InstallShield adds a property action to the **Actions** explorer.
3. Enter a new name, or right-click it later and click **Rename** to assign it a new name. Use a name that helps you distinguish this action from other actions in your project.
4. Configure the action's settings as needed.

Once you have added the action to your project, schedule it as needed. For more information, see Scheduling a Suite/Advanced UI Action.

Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation

Project *• This information applies to Suite/Advanced UI projects.*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add to your Suite/Advanced UI project an action that calls InstallScript code at run time. These Suite/Advanced UI–based InstallScript actions function much like InstallScript custom actions in Basic MSI projects.

The first step in adding an InstallScript action to a Suite/Advanced UI project is to write the InstallScript code that you want to run from your installation. You can use the InstallScript view in your project to do this. Once you have written the code, you can add an InstallScript action through the Events view of your project.

Task ***To add an action that runs InstallScript code in a Suite/Advanced UI installation:***

1. Add the InstallScript code to your project:
   a. In the View List under **Behavior and Logic**, click **InstallScript**.
   b. In the **InstallScript** explorer, right-click the **Files** node and then click **New Script File**. InstallShield adds a new file called Setup.rul under the **Files** node.
   c. Select the new .rul file, and then in the right pane, write the code that you want to launch at run time.

      To have your Suite/Advanced UI action call an InstallScript function, you must prototype the function with the **export** keyword; for example:

      ```
      export prototype MyFunction(OBJECT);
      ```
2. Add an InstallScript action that calls your InstallScript code:
   a. In the View List under **Behavior and Logic**, click **Events**.
   b. Right-click the **Actions** explorer and then click **New InstallScript**. InstallShield adds an InstallScript action to the **Actions** explorer.
   c. Enter a new name, or right-click it later and click **Rename** to assign it a new name. Use a name that helps you distinguish this action from other actions in your project.
   d. Configure the action's settings as needed.

Once you have added the action to your project, schedule it as needed. For more information, see Scheduling a Suite/Advanced UI Action.

Tips for Writing InstallScript Code for Suite/Advanced UI Actions

Several InstallScript functions wrap the ISuiteExtension object that is passed to an InstallScript action entry point. These InstallScript functions enable your InstallScript actions to interact with the running Suite/Advanced installation.

- **SuiteFormatString**
- **SuiteGetProperty**
- **SuiteLogInfo**
- **SuiteResolveString**
- **SuiteSetProperty**

To learn more about these functions, see Suite/Advanced UI and Advanced UI Interaction Functions.

Note the following details about the use of InstallScript functions that are called from actions in Suite/Advanced UI installations:

- No InstallScript UI–related functions are available for use in a Suite/Advanced UI project; attempting to use any results in either the script failing to compile or, if the script can be compiled, an error is returned at run time. It is highly recommended that any UI that needs to be displayed by an InstallScript action instead be deferred to the UI of the Suite/Advanced UI installation.
- An InstallScript action can read and write Suite/Advanced UI properties and use them as a sort of communication mechanism with the Suite UI. (Suite/Advanced UI actions are similar to Basic MSI custom actions and have no access to the UI.) For a list of built-in properties, see Advanced UI and Suite/Advanced UI Property Reference.
- InstallScript run-time path variables (PROGRAMFILES, WINDIR, etc.) are not available. These variables normally are implemented through a concept in the script engine known as *bound variables*, wherein the variables themselves are associated directly to textsub name-value mappings. The textsubs that correspond with these variables are still available; for example, <PROGRAMFILES> can be resolved through the InstallScript functions **TextSubGetValue** or **TextSubSubstitute**. Path-type Suite/Advanced UI properties are also defined through the Suite/Advanced UI engine and can be read with **SuiteGetProperty**.
- The szStatusText member of the LAAW_PARAMETERS structured variable is not available for use in InstallScript actions that are called in Suite/Advanced UI installations.

You can debug the InstallScript actions in your Suite/Advanced UI installation with the InstallScript Debugger. To learn more, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.

Working with a Managed-Code Action in a Suite/Advanced UI Installation

Project *• This information applies to Suite/Advanced UI projects.*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you add to your Suite/Advanced UI project an action that calls a public method in a managed assembly that is written in managed code such as Visual Basic .NET or C# at run time.

Creating a Managed Assembly for a Managed-Code Action

The first step in adding a managed-code action to a Suite/Advanced UI project is to create the managed assembly. You can create the managed assembly with any recent version of .NET languages such as Visual Basic .NET or C#.

The Suite/Advanced UI engine requires that each managed-code action have one entry point. An entry point in this case is either a static method, or an instance method on a class that can be constructed without any parameters.

To specify the entry point, use the Class setting and the Method setting for the action in the Events view.

The Suite/Advanced UI installation calls the entry point when an action in the installation is scheduled to run. Your entry-point method should return the number zero (0) to indicate that the action finished successfully. If the action fails, it should return a non-zero value. The error values that are returned are the same as those defined for return in Windows Installer custom actions.

The interface that is passed to the action's entry point implements the ISuiteExtension interface, which is passed as an object. The return type should be a 32-bit integer type.

To work with a Suite/Advanced UI extension interface, you can include code such as the following when you are writing C# code, for example, in Visual Studio:

```
[Guid("BAFAEAED-08C6-4679-B94E-487A4D89DE63")]
[TypeLibType(4288)]
public interface ISuiteExtension
    {
        [DispId(1)]
        string get_Attribute(string bstrName);
        [DispId(2)]
        void LogInfo(string bstr);
        [DispId(3)]
        string get_Property(string bstrName);
        [DispId(3)]
        void set_Property(string bstrName, string bstrValue);
        [DispId(4)]
        string FormatProperty(string bstrValue);
        [DispId(5)]
        string ResolveString(string bstrStringId);
    }
```

You can use the following methods in your managed code:

- **LogInfo**—This method writes any information that is needed to the Suite/Advanced UI debug log, making it available for debugging or other informational purposes. The bstr parameter contains the string to be written to the log.
- **get_Property**—This method retrieves the value of any property that is currently defined in the running Suite/Advanced UI installation. Properties that are not defined return an empty value. The bstrName parameter specifies the name of the property whose value is being obtained. The value for the property is returned in the bstrValue parameter.
- **set_Property**—This method sets the value of a new property or changes the value of an existing property in the currently running Suite/Advanced UI installation. Passing an empty value effectively deletes the property. The bstrName parameter specifies the name of the property to set. The bstrValue parameter specifies the value to set for the specified property.
- **FormatProperty**—This method resolves embedded formatted expressions in the string that is provided in the bstrValue parameter. The formatted value is returned in the bstrReturnValue parameter. To learn about the

syntax that is available for formatted expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

- **ResolveString**—This method resolves a Suite/Advanced UI string identifier into the corresponding string value for the currently running Suite/Advanced UI installation in the currently selected UI language. The bstrStringId parameter specifies the string identifier to resolve, and the resolved string is returned in bstrReturnValue. If no such string identifier exists, the returned string is empty.

The following sample C# code corresponds with a Class setting value of TestCLRAction.TestCLRClass and a Method setting value of TestCLRMethod:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Runtime.InteropServices;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using Microsoft.WindowsAzure.Common;
using Microsoft.WindowsAzure.Management;
using Microsoft.WindowsAzure.Management.WebSites;
using Microsoft.WindowsAzure.Management.WebSites.Models;
using Microsoft.WindowsAzure;
using System.Windows.Forms;

namespace TestCLRAction
{
    [Guid("BAFAEAED-08C6-4679-B94E-487A4D89DE63")]
    [TypeLibType(4288)]
    public interface ISuiteExtension
    {
        [DispId(1)]
        string get_Attribute(string bstrName);
        [DispId(2)]
        void LogInfo(string bstr);
        [DispId(3)]
        string get_Property(string bstrName);
        [DispId(3)]
        void set_Property(string bstrName, string bstrValue);
        [DispId(4)]
        string FormatProperty(string bstrValue);
        [DispId(5)]
        string ResolveString(string bstrStringId);
    }

    public class TestCLRClass
    {
        const UInt32 ERROR_INSTALL_FAILURE = 1603;
        const UInt32 ERROR_SUCCESS = 0;

        ISuiteExtension GetExtensionFromDispatch(object pDispatch)
        {
            if (Marshal.IsComObject(pDispatch) == false)
                throw new System.ContextMarshalException("Invalid
                dispatch object passed to CLR method");

            return (ISuiteExtension)pDispatch;
```

```
        }
        public UInt32 TestCLRMethod(object pDispatch)
        {
            try
            {
                ISuiteExtension suiteExtension = GetExtensionFromDispatch(pDispatch);
                suiteExtension.set_Property("TESTPROP", "hello, world");
            }
            catch (System.ContextMarshalException)
            {
                return ERROR_INSTALL_FAILURE;
            }

            return ERROR_SUCCESS;
        }
    }
```

Note that although the Suite/Advanced UI installation allows any action to be performed, actions are run with the privileges that they are configured to have. Some actions may require administrator privileges (such as reading IIS 7.x configuration data). In such cases, in order for the action to access the required data successfully, the action would need to be configured to require administrator privileges; that is, Yes would need to be selected for the Requires Administrative Privileges setting for the action in the Events view. As an alternative, the Suite/Advanced UI installation's **Setup.exe** file could be run with administrator privileges, or it would need to include an administrator manifest. However, privileges should be elevated for as short of a duration as possible.

Adding a Managed-Code Action



Task ***To add an action that calls a public method in a managed assembly during a Suite/Advanced UI installation:***

1. In the View List under **Behavior and Logic**, click **Events**.
2. Right-click the **Actions** explorer and then click **New Managed Code**. InstallShield adds a managed-code action to the **Actions** explorer.
3. Enter a new name, or right-click it later and click **Rename** to assign it a new name. Use a name that helps you distinguish this action from other actions in your project.
4. Configure the action's settings as needed.

Once you have added the action to your project, schedule it as needed. For more information, see Scheduling a Suite/Advanced UI Action.

Run-Time Requirements for a Managed-Code Action

You can use the Suite/Advanced UI property **IS_CLR_VERSION** to identify a semicolon-delimited list of .NET Framework versions that the action should attempt to load to run your managed code. To learn more about the **IS_CLR_VERSION** property, see Advanced UI and Suite/Advanced UI Property Reference.

If the managed-code action has any dependencies, add these to your project through the Support Files view. At run time, the Suite/Advanced UI engine specifies to the .NET Framework that the dependencies are in [SETUPSUPPORTDIR].

Adding an Action to a Suite/Advanced UI Project

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Task

To add an action to a Suite/Advanced UI project:

1. In the View List under **Behavior and Logic**, click **Events**.
2. Right-click the **Actions** explorer and then click the type of action that you want to add.

   If you are creating a PowerShell action, the **Open** dialog box opens and enables you select the PowerShell script file (.ps1) that you want to use.
3. Optionally rename the new action by selecting it, pressing the F2 key, and typing the new name.
4. Configure the action's settings.

Note that if the action cannot be launched or loaded at run time, the installation is aborted.

Configuring a Suite/Advanced UI Action's Settings

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Task

To configure an action in a Suite/Advanced UI project:

1. In the View List under **Behavior and Logic**, click **Events**.
2. In the **Actions** explorer, click the type of action that you want to configure.
3. Configure the settings in the grid on the right.

Note that if the action cannot be launched or loaded at run time, the installation is aborted.

Scheduling a Suite/Advanced UI Action

Project *• This information applies to Suite/Advanced UI projects.*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Once you have added an action to your project, you can schedule when at run time you want it be launched. You can also define one or more conditions that the Suite/Advanced UI engine should evaluate before launching the action.

InstallShield has support for scheduling actions in different types of places:

- You can schedule an action to run during one or more of the built-in events that the Suite/Advanced UI manages. For example, shortly after a Suite/Advanced UI installation is loaded, the Suite/Advanced UI engine triggers the execution of an event called OnBegin. This event specifies the instructions that need to be carried out when the OnBegin event occurs.
- You can schedule an action to run before or after the Suite/Advanced UI engine installs, removes, modifies, or repairs a package.
- You can associate an action with an element of the wizard interface, such as a wizard page or a control on one of the wizard pages.

For more detailed information, see Types of Events in a Suite/Advanced UI Installation.

Scheduling an Action During an Event

The events and actions that are listed under the Events explorer in the Events view are organized by chronological order, according to when they are launched at run time. When you add an action to an event, you specify when the action should be launched by adding it to the appropriate event.

Task

To schedule an action during an event:

1. In the View List under **Behavior and Logic**, click **Events**.
2. In the **Events** explorer, right-click the event or action that you want your action to follow, point to **Add Existing Action**, and then click the name of the action that you are scheduling. InstallShield adds the action to the **Events** explorer.
3. Configure the settings in the grid on the right. One of the settings lets you build conditional statements that control whether the action is launched. For details on how to define conditions, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.

   To indicate that an action should run only during certain modes, use the Suite/Advanced UI property **ISInstallMode** in a Property Comparison condition. For more information on this and other built-in properties, see Advanced UI and Suite/Advanced UI Property Reference.

Tip *• The Events view supports drag-and-drop editing:*

- *To sequence a new action, you can drag it from the Actions explorer to the appropriate event under the Events explorer. When you drop it, drop it onto the item that should be directly before it in the sequence.*
- *To move an action to a different point in an event (or from one event to another), drag it from the old location and drop it onto the item that should be directly before it in the sequence.*
- *To copy an action from one event to another, press and hold CTRL while dragging the action from one event to another event, and drop it onto the action or event that should be directly before it.*

Scheduling an Action Before or After the Suite/Advanced UI Engine Installs, Removes, Modifies, or Repairs a Package



Task ***To schedule an action before or after a package is installed, removed, modified, or repaired:***

1. In the View List under **Organization**, click **Packages**.
2. In the **Packages** explorer, click the package that you want to be associated with the action.
3. On the **Common** tab, expand the **Events** area if it is not already expanded.
4. Do one of the following:
   - To schedule an action before the package is installed, removed, modified, or repaired: In the **Package Configuring** setting, point to the **New Action** button, and then click the name of the action that you are scheduling. InstallShield adds an **Action** subsetting and sets its value to the name of the action that you selected.
   - To schedule an action after the package is installed, removed, modified, or repaired: In the **Package Configured** setting, point to the **New Action** button, and then click the name of the action that you are scheduling. InstallShield adds an **Action** subsetting and sets its value to the name of the action that you selected.
5. Optionally, to build a conditional statement so that you can further control when the action is launched, click the **Add a condition** button. For details on how to define conditions, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.

   To indicate that an action should be run only if a specific package in the installation is running in a particular state, create a Package Operation condition.

   To indicate that an action should run only during certain modes, use the Suite/Advanced UI property **ISInstallMode** in a Property Comparison condition. For more information on this and other built-in properties, see Advanced UI and Suite/Advanced UI Property Reference.

Adding an Action to an Element in the Wizard Interface

InstallShield lets you define actions that are launched when a wizard page or secondary window is opened or closed. In addition, InstallShield lets you define actions that an end user triggers when using some of the various wizard interface controls—for example, when an end user clicks a button. To learn how to schedule an action through the wizard interface, see Configuring an Action for an Element in the Wizard Interface.

Types of Events in a Suite/Advanced UI Installation

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Suite/Advanced UI installations are driven by the Suite/Advanced UI installation, which manages a series of events in a specific order. These events run actions that you have added to your project.

The following table describes each of the events that are listed in the Events view of a Suite/Advanced UI installation:

Table 5-3 • Events That Are Listed in the Events View

Event Name	Description
OnBegin	The OnBegin event is the first event in a Suite/Advanced UI installation. Actions that are scheduled for this event are executed during all modes (install, maintenance, repair, etc.) unless the conditional statement that is defined for the action evaluates as false. To indicate that an action should run only during certain modes, use the Suite/Advanced UI property **ISInstallMode** in a Property Comparison condition. For more information on this and other built-in properties, see Advanced UI and Suite/Advanced UI Property Reference. Actions that are scheduled during OnBegin run before the Suite/Advanced UI installation displays the wizard interface. A long-running action during this event could result in a long delay between the moment when an end user launches the installation and the point at which the wizard interface is first shown. This could lead the end user to think that the installation was not launched properly. Thus, you may want to avoid scheduling long-running actions during OnBegin. Note that you also may want to avoid using an action that displays its own user interface when launched. The wizard interface of the Suite/Advanced UI installation can appear behind other open windows in this scenario, possibly confusing an end user.
OnResuming	The OnResuming event launches its actions when a Suite/Advanced UI installation is resuming after the target machine has been restarted.
OnStaging	The OnStaging event launches its actions immediately before the Suite/Advanced UI installation is staged; that is, before the Suite/Advanced UI installation downloads its package (if applicable), extracts its packages from the **Setup.exe** file (if applicable), and copies its packages to a directory that the end user specifies on the BrowseStageFolder wizard page or by setting the **ISRootStagePath** property through the command line.

Table 5-3 • Events That Are Listed in the Events View (cont.)

Event Name	Description
OnStaged	The OnStaged event launches its actions after the Suite/Advanced UI installation has downloaded its packages (if applicable), extracted them from the **Setup.exe** file, and copied them to the staging directory.
OnPackagesConfiguring	The PackagesConfiguring event launches its actions immediately before the packages in the Suite/Advanced UI installation are being installed, modified, repaired, or removed.
OnPackagesConfigured	The OnPackagesConfigured event launches its actions when the packages in the Suite/Advanced UI installation have been installed, modified, repaired, or removed.
OnRebooting	The OnRebooting event launches its actions immediately before the Suite/Advanced UI installation triggers a restart of the machine. Note that actions that are scheduled during OnRebooting should run for a brief period of time. Otherwise, the actions may fail to complete if a reboot is initiated by some prior action and Windows shuts down the Suite/Advanced UI process before the reboot events complete.
OnEnd	The OnEnd event launches its actions immediately before the end of a successful Suite/Advanced UI installation. If the end user cancels the Suite/Advanced UI installation or the installation encounters an error, the OnEnd event does not run.

Note that the Suite/Advanced UI engine does not maintain states to track which events have run or not run. Therefore, you may need to define Property Comparison conditions for actions to prevent the actions from being rerun after the Suite/Advanced UI installation resumes because of a reboot. You can use the ISOnRebooted property in these conditions; if the Suite/Advanced UI installation is resuming after a reboot, this property is set to True.

If you define a Feature Operation condition or a Package Operation condition for an action, the earliest event for which you can schedule the action is OnStaging.

Defining Exit Conditions for an Advanced UI or Suite/Advanced UI Installation



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you specify one or more exit error messages that you want the Advanced UI or Suite/Advanced UI installation to display under various conditions before ending the installation. For example, if your Advanced UI or Suite/Advanced UI installation requires Windows Vista or later, you could set up an exit message to inform end users about the requirement. You would also define a condition for that exit message to evaluate whether target systems have earlier versions of Windows. If the condition that you define is true when an end user launches the Advanced UI or Suite/Advanced UI installation, the Advanced UI or Suite/Advanced UI installation displays the error message. When end users dismiss the error message, the Advanced UI or Suite/Advanced UI installation ends.

Task ***To define an exit condition and message for your Advanced UI or Suite/Advanced UI installation:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Exit Conditions** setting, click the **New Condition** button. InstallShield adds exit message subsettings under the **Exit Conditions** setting.
3. In the **Exit Message** setting, enter the error message that you want the Advanced UI or Suite/Advanced UI installation to display at run time if the condition that you are configuring is true.
4. Expand the **Exit Message** setting, and use its subsetting to define the conditional statement that you want the Advanced UI or Suite/Advanced UI installation to evaluate at run time.
    - In the first subsetting, use the list to select the option that you want to use for the condition or group of conditions that you are configuring. Available options are:
        - **Any**—An Any condition group operates like a logical OR operation. If any of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to true. If none of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to false.
        - **All**—An All condition group operates like a logical AND operation. All of the conditions that are in the All group must evaluate to true in order for the condition group to evaluate to true.
        - **None**—A None condition group operates like a logical NOR operation. If none of the conditions that are in the None group evaluate to true, the entire condition group evaluates to true. If any of the conditions that are in the None condition group evaluate to true, the entire condition group evaluates to false. Note that if the None condition group consists of only one conditional statement, it operates like a logical NOT operation.
    - To add a subcondition group, click the **New Condition** button in this setting and then select **Any**, **All**, or **None**, as needed. InstallShield adds a new row that you can use to define the condition.
    - To add a condition, click the **New Condition** button in this setting and then select one of the available condition types. InstallShield adds a new row that you can use to define the condition.

For more details on configuring conditions, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.

For descriptions of each of the condition settings, see Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects.

Triggering Install Mode or Maintenance Mode of an Advanced UI or Suite/Advanced UI Installation



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

An Advanced UI or Suite/Advanced UI installation runs in one of the following modes:

- Install mode, in which the installation behaves as a first-time installation
- Maintenance mode, in which the installation displays a wizard page that lets end users select whether they want to remove, repair, or modify the product

InstallShield creates install and maintenance mode conditions automatically in Advanced UI and Suite/Advanced UI projects. The mode conditions are not available for edit within InstallShield; they are shown only in the project file (.issuite). These mode conditions determine whether an Advanced UI or Suite/Advanced UI installation runs in first-time installation mode or in maintenance mode.

Install Mode Condition

The install mode condition is based on the following factors:

- **Detection conditions**—Part of the install mode condition consists of the detection conditions of all of the primary packages in the Suite/Advanced UI project (or, in an Advanced UI project, the one primary package). If none of the primary packages' detection conditions evaluate as true (that is, if none of the primary packages are already installed), the detection condition part of the install mode condition is true. If one or more of the primary packages' detection conditions evaluate as true (that is, at least one of the primary packages is already installed), the detection condition part of the install mode condition is false.
- **Suite Installed condition**—The other part of the install mode condition consists of a Suite Installed condition, which may trigger the installation to run in first-time installation mode if the same version of the Advanced UI or Suite/Advanced UI installation is absent from the target system. The Suite Installed part of the install mode condition compares the Suite GUID and the version of the Advanced UI or Suite/Advanced UI installation with, if applicable, those of the Advanced UI or Suite/Advanced UI installation that is installed on the target system. If these values are different, the Suite Installed part of the install mode condition suggests that the Advanced UI or Suite/Advanced UI installation is absent, so a first-time installation may result. For more details about the Suite Installed type of condition check, see Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed.

If all of the primary packages' detection conditions evaluate as false, or if the Suite Installed condition indicates that the Advanced UI or Suite/Advanced UI installation is not already installed, the install mode condition is true, and the installation runs as a first-time installation.

Maintenance Mode Condition

The maintenance mode condition is based on the detection conditions of all of the primary packages in the Suite/Advanced UI project (or, in an Advanced UI project, the one primary package). If any of the primary packages' detection conditions evaluate as true (that is, if one or more of the primary packages are already installed), the mode condition is true, and the installation runs in maintenance mode.

Special Considerations for Suite/Advanced UI Projects with Web Deploy Packages

Web Deploy packages cannot be detected. These types of packages are typically intended to behave similarly to first-time installations or to install over existing earlier or identical versions. As a side effect, Web Deploy packages do not handle uninstallation. Thus, Suite/Advanced UI installations do not offer maintenance for this type of package. It is recommended that you avoid creating a Suite/Advanced UI installation that combines one or more Web Deploy packages with one or more traditional packages (that do offer maintenance) as primary packages.

Add or Remove Program Entries for an Advanced UI or Suite/Advanced UI Installation



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Advanced UI and Suite/Advanced UI projects let you specify whether you want to have an entry in Add or Remove Programs for your Advanced UI or Suite/Advanced UI installation. This entry lets end users perform maintenance for your Advanced UI or Suite/Advanced UI installation, modifying or removing if needed. The General Information view in an Advanced UI or Suite/Advanced UI project has a Show Add or Remove Programs Entry setting that lets you indicate the appropriate behavior.

If you want to show only a single entry for the entire Advanced UI or Suite/Advanced UI installation, ensure that you hide the entries from the packages that you include in the Advanced UI or Suite/Advanced UI project.

For an .msi package, for example, you could set the value of the Windows Installer property **ARPSYSTEMCOMPONENT** to 1 through the command-line settings for the package in the Packages view.

Note that the Show Add or Remove Programs Entry setting is not the only factor that determines whether the Advanced UI or Suite/Advanced UI engine creates an Add or Remove Programs entry for your Advanced UI or Suite/Advanced UI product. The maintenance mode condition also determines whether this occurs. The maintenance mode condition is a combination of all of the detection conditions for all of the primary packages in the installation.

If the detection conditions for all of the primary packages in an Advanced UI or Suite/Advanced UI installation evaluate to false after a first-time installation, the Advanced UI or Suite/Advanced UI engine does not add an Add or Remove Program entry for the Advanced UI or Suite/Advanced UI product. If the detection conditions for all of the primary packages in an Advanced UI or Suite/Advanced UI installation evaluate to true after the Advanced UI or Suite/Advanced UI product is removed, the Advanced UI or Suite/Advanced UI engine leaves any existing Add or Remove Program entry for the Advanced UI or Suite/Advanced UI product.

Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are configuring a package in the Packages view of an Advanced UI or Suite/Advanced UI project, you can use the command line and silent command line settings to specify the command lines that you want the Advanced UI or Suite/Advanced UI installation to use when launching the package. These settings are available as subsettings in the Operation area of the grid in the Packages view.

If the Advanced UI or Suite/Advanced UI installation is run with a user interface, the Advanced UI or Suite/Advanced UI **Setup.exe** file uses the command line that you enter in the MSI Command Line setting, the MSP Command Line setting, the Command Line setting, or the EXE Command Line setting (depending on the package type) to launch the package. If the installation is run silently (without a user interface), the Advanced UI or Suite/Advanced UI **Setup.exe** file uses the command line that you enter in the MSI Silent Command Line setting, the MSP Silent Command Line setting, the Silent Command Line setting, or the EXE Silent Command Line setting to launch the package.

When you are entering values in these settings, refer to the following background information, depending on which type of package you are configuring.

.msi Package

If you are configuring the Install and Remove operations for an .msi package, the only type of command-line parameters that you should enter are Windows Installer properties. An Advanced UI or Suite/Advanced UI installation uses MsiInstallProduct to launch an .msi package in install or remove mode, and this function accepts only Windows Installer properties as command lines.

If you are configuring the Repair and Modify operations for this type of package, the only type of command-line parameters that you should enter are Windows Installer feature properties:

- **ADDDEFAULT**

- **ADDLOCAL**
- **ADDSOURCE**
- **ADVERTISE**
- **COMPADDDEFAULT**
- **COMPADDLOCAL**
- **COMPADDSOURCE**
- **FILEADDDEFAULT**
- **FILEADDLOCAL**
- **FILEADDSOURCE**
- **MSIDISABLELUAPATCHING**
- **MsiPatchRemovalList**
- **MSIRESTARTMANAGERCONTROL**
- **MSIDISABLERMRESTART**
- **MSIRMSHUTDOWN**
- **MSIPATCHREMOVE**
- **PATCH**
- **REINSTALL**
- **REINSTALLMODE**
- **REMOVE**

An Advanced UI or Suite/Advanced UI installation uses MsiConfigureProductEx to launch an .msi package in repair or modify mode, and this function accepts only Windows Installer feature properties.

If you are using a Windows Installer property for a package's MSI Command Line settings and the MSI Silent Command Line settings, use the following format:

```
MYPROPERTYNAME=MyPropertyValue
```

The /l command-line option cannot be passed to the package to generate a log file. If you want to log the installation, consider enabling logging for the package in the Packages view. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation. As an alternative, you can use either the logging system policy, or the MsiLogging property to log the installation.

Note that an Advanced UI or Suite/Advanced UI installation always launches .msi packages silently. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the .msi package.

.msp Package

If you are configuring the Install operation for an .msp package, the only type of command-line parameters that you should enter are Windows Installer properties. An Advanced UI or Suite/Advanced UI installation uses MsiApplyPatch to apply an .msp package to a target system, and this function accepts only Windows Installer properties as command lines.

To update all of the features through an .msp package, you would enter command-line properties such as the following:

```
REINSTALLMODE=vomus REINSTALL=ALL
```

To update only certain features that are included in the .msp package, you would set **REINSTALL** to a comma-separated list of features that you want to be updated:

```
REINSTALLMODE=vomus REINSTALL=Feature1,Feature3,Feature5
```

Note that an Advanced UI or Suite/Advanced UI installation always launches .msp packages silently. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the .msp package.

InstallScript Package

If you are configuring the Install and Modify operations for an InstallScript package, you can use the Advanced UI and Suite/Advanced UI properties **ISFeatureInstall** and **ISFeatureRemove** in your command line. The InstallScript function FeatureConfigureFeaturesFromSuite uses these properties to set feature states for the InstallScript package.

You can set these properties to a comma-delimited list of feature names as follows in the Command Line setting and the Silent Command Line setting:

```
ISFeatureInstall=Feature1,Feature2 ISFeatureRemove=Feature3
```

In the above example, **Feature1** and **Feature2** are selected to be installed; **Feature3** is selected to be removed, if it is present.

If any features are used in the values of both properties, the features that are set for the **ISFeatureRemove** property supersede the **ISFeatureInstall** property.

Note that Advanced UI and Suite/Advanced UI installations launch InstallScript packages silently by default. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the InstallScript package, and it is not necessary to use a silent response file.

.exe Package

If you are configuring the Install, Remove, Repair, and Modify operations for an .exe package, you can enter any command-line parameters that are supported by the .exe file.

If all of the customization can be done without end-user intervention, consider including the command-line parameter for running the .exe package silently.

Silently Running the Setup.exe File that Was Built in InstallShield for a Windows Installer–Based Installation

To have the Advanced UI or Suite/Advanced UI installation launch a Windows Installer–based **Setup.exe** installation silently, use the following syntax in the EXE Command Line settings and the EXE Silent Command Line settings:

```
Setup.exe /s /v"/qn"
```

Silently Running the Update.exe File that Was Built in InstallShield for a Windows Installer–Based Patch

To have the Advanced UI or Suite/Advanced UI installation launch a Windows Installer–based **Update.exe** patch silently, enter the following in the EXE Command Line settings and the EXE Silent Command Line settings:

```
Update.exe /s /v"/qn"
```

Silently Running the Setup Launcher File for an InstallScript Installation

To have the Advanced UI or Suite/Advanced UI installation launch an InstallScript **Setup.exe** installation silently, create a silent response file, and add it as a support file in the Support Files view of your Advanced UI or Suite/Advanced UI project. In the EXE Command Line setting and the EXE Silent Command Line setting of the Install operation, use the following syntax:

```
Setup.exe /s /f1"[SETUPSUPPORTDIR]\Setup.iss"
```

Instead of launching the original **Setup.exe** file for uninstallation silently, you can create a silent response file for the uninstallation, and add it as a support file in the Support Files view of your Advanced UI or Suite/Advanced UI project. In the EXE Command Line setting and the EXE Silent Command Line setting of the Remove operation, use the following syntax:

```
"[ProgramFilesFolder]InstallShield Installation Information\{PRODUCT-GUID-HERE}\Setup.exe" /s /
f1"[SETUPSUPPORTDIR]\Uninstall.iss" -remove_only -runfromtemp
```

Using Advanced UI and Suite/Advanced UI Formatted Expressions to Dynamically Configure Command Lines



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield lets you use Advanced UI and Suite/Advanced UI formatted expressions to dynamically configure the command lines that the Advanced UI or Suite/Advanced UI installation uses to launch the Advanced UI or Suite/Advanced UI packages. The formatted expressions can consist of property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions.

In some scenarios, you can use a single Advanced UI or Suite/Advanced UI formatted expression to configure most or all of your packages; in other scenarios, each package requires a different Advanced UI or Suite/Advanced UI formatted expression.

To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

The following sample scenarios explain how to use an Advanced UI or Suite/Advanced UI property as your formatted expression to pass values to packages in the installation.

Using an Advanced UI or Suite/Advanced UI Property to Pass the Same Value to Packages in the Advanced UI or Suite/Advanced UI Installation

When you are creating an Advanced UI or Suite/Advanced UI installation, you may need to use an Advanced UI or Suite/Advanced UI property to set a Windows Installer property to change how each .msi package is installed. For example, if you are creating a multilanguage installation, you may need to select a language transform that controls which language is installed. In this particular example, the language (and hence, the property value) would probably be the same for each package in your Advanced UI or Suite/Advanced UI project.

In an Advanced UI project, the Advanced UI or Suite/Advanced UI property **ISSelectedLanguage** property identifies the language that is used for the Advanced UI or Suite/Advanced UI installation. If you configure your release to allow end users to select the language, the InstallationLanguage wizard page prompts end users for the language; the installation sets the **ISSelectedLanguage** property to a string that contains the decimal language identifier of the language that they selected.

In order to run the .msi packages in the Advanced UI or Suite/Advanced UI project in the appropriate language, include the following in the value that you enter for the command line settings for each .msi package:

```
TRANSFORMS="[ISPREREQDIR]\[ISSelectedLanguage].mst"
```

The .mst file should be in the same folder as the .msi package.

Use the following syntax for your command line settings of Basic MSI .exe packages:

```
/L[ISSelectedLanguage]
```

Using an Advanced UI or Suite/Advanced UI Property to Pass Different Values to Packages in the Advanced UI or Suite/Advanced UI Installation

You may want to use an Advanced UI or Suite/Advanced UI property to pass different property values to each or some packages in your Advanced UI or Suite/Advanced UI installation. For example, you may want to allow end users to override **INSTALLDIR** to different locations, one for each package. In this example, the value of **INSTALLDIR** could be different for each package.

When you add the predefined BrowseFolder wizard page to your project for one of the packages in your Advanced UI or Suite/Advanced UI project, InstallShield automatically adds the following to the package's command line settings for the install operation in the Packages view:

```
INSTALLDIR="[ISInstallDir_PackageDisplayName]"
```

You can override that command line if appropriate. If the package is an .msi package, that command line sets the package's INSTALLDIR value to the path that the end user selected on the BrowseFolder wizard page for that package.

If the package is a Basic MSI or InstallScript MSI .exe package, change the default command lines to use this syntax:

```
/v"INSTALLDIR=\"[ISInstallDir_PackageDisplayName]\""
```

Defining the End-User Interface of an Advanced UI or Suite/Advanced UI Installation



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The elements of the wizard interface of an Advanced UI or Suite/Advanced UI installation consist of wizard pages and secondary windows (also known as pop-up windows). Advanced UI and Suite/Advanced UI projects contain a number of standard predefined wizard pages and secondary windows. You can add or delete pages and windows, as well as customize the layout and behavior of them.

Note that the Advanced UI or Suite/Advanced UI setup launcher automatically suppresses the user interface of .msi, .msp, and InstallScript package formats, in favor of the wizard interface of the Advanced UI or Suite/Advanced UI installation. If an Advanced UI or Suite/Advanced UI project contains an .exe package, you must manually suppress its user interface and have the Advanced UI or Suite/Advanced UI installation launch it silently. Otherwise, at run time, the installation shows the Advanced UI or Suite/Advanced UI wizard pages and the separate user interface of the .exe package, presenting two disparate user interfaces. For more information, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

To learn more about defining the end-user interface of an Advanced UI or Suite/Advanced UI installation, see Working with the Wizard Interface.

Referring to Feature States and Other Feature Data in the UI of an Advanced UI or Suite/Advanced UI Project

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

At run time during an Advanced UI or Suite/Advanced installation, you may need to trigger specific UI behavior that depends on feature-related information. Following are examples of how you may want to use this information:

- If a particular feature is not selected for installation, you can skip a special corresponding wizard page.
- If a specific feature is already installed, you can disable a check box on one of the wizard pages.
- You can create an extension condition DLL to define a custom condition for a feature-related check that you want the installation to perform at run time.

- You can configure the installation to display a custom feature tree wizard page that shows how much space on the target system each feature requires.

Following are various feature-related pseudo properties that you can check at run time during an Advanced UI or Suite/Advanced UI installation. In each case, the *name* portion should be replaced with the name of the feature.

Table 5-4 • Feature-Related Pseudo Properties for the UI of an Advanced UI or Suite/Advanced UI Installation

Pseudo Property	Type	Description
FEATURE[*name*].actionState	Read-write	This indicates the upcoming action for the specified feature. Available values are: • **install**—This state indicates that the feature is set to be installed. This state is valid only when the feature is not already installed. • **remove**—This state indicates that the feature is set to be removed. This state is valid only when the feature is already installed. • **null (empty string)**—An empty string state requests no action. It is always valid.
FEATURE[*name*].installState	Read-only	This indicates the current state of the specified feature. Available values are: • **0**—The feature is currently absent from the target system. • **1**—The feature is partially available on the target system; that is, only some of the packages in this feature are present. • **2**—Only some of the feature's child features are present on the target system. • **3**—The feature is currently present on the target system.
FEATURE[*name*].displayName	Read-only	This indicates the name that is displayed for the specified feature on the appropriate wizard page at run time.
FEATURE[*name*].description	Read-only	This indicates the description of the specified feature that is displayed on a wizard page at run time.
FEATURE[*name*].cost	Read-only	This indicates how much space the specified feature requires on the target system.

You can set the read-write pseudo property FEATURE[name].actionState through an extension condition, for example, or through the Action setting for a control in the wizard interface.

You can read any of those pseudo properties through an extension condition or through in binding conditions for the Visible or Enabled settings for a control in the wizard interface.

Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

In various areas of Advanced UI and Suite/Advanced UI projects, you can embed formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. Support for the following types of formatted expressions is available.

- **Properties**—To resolve the value of a property, surround the property name in square brackets. For example, **[MYPROPERTY]** resolves to the value of the MYPROPERTY property at run time. If the property is not defined, the expression resolves to an empty string.
- **Recursive properties**—To resolve the resolved value of a property, surround a property name with an additional set of square brackets. For example, **[[PROPERTY1]]** resolves to the value of the property whose name is stored in **PROPERTY1**; if **PROPERTY1** holds **PROPERTY2**, the resolved value is the value that is stored in **PROPERTY2**.
- **Escaped characters**—To include a special character in an expression (such as literal parentheses, commas, or square brackets), precede the special character with a backslash and enclose the resulting string in square brackets. For example, an entry such as **[\[]Text[\]]** resolves as **[Text]**.
- **Null character**—To embed a null character in an expression, enclose a tilde in square brackets; that is, **[~]**.
- **Environment variables**—To use the value of an environment variable, precede the name of the environment variable with a percent sign and enclose the resulting string in square brackets. For example, **[%PATH]** resolves as the value of the PATH environment variable.
- **Object expressions**—To search target systems for information about a file, a registry entry, the operating system, or other items, use an object expression. This enables you to dynamically configure many of the Advanced UI or Suite/Advanced UI settings at run time based on target system–specific conditions. Object expressions use the convention **[@Object(Parameters, ...).Property(Parameters, ...)]**. For details on how to write object expressions, see Writing Object Expressions in Advanced UI and Suite/Advanced UI Projects to Search Target Systems.

Note that you can embed object expressions within other formatted expressions, such as within property expressions. You can also embed property expressions and other formatted expressions within a parameter of an object expression. To learn more, see Writing Object Expressions in Advanced UI and Suite/Advanced UI Projects to Search Target Systems.

Syntax errors, failures of objects in object expressions, and invalid formatted expressions evaluate to an empty string at run time. In some cases, information about the failure to parse or evaluate a formatted expression is available in the debug log file of the Advanced UI or Suite/Advanced UI installation.

Areas of Advanced UI and Suite/Advanced UI Projects That Accept Formatted Expressions

Many settings in various areas of Advanced UI and Suite/Advanced UI projects accept formatted expressions. You can also use formatted expressions in some types of actions. The following areas in InstallShield are examples in which you can specify formatted expressions.

- In the Packages view: settings in which you enter command lines that should be used to launch a package's file.

  For more information, see Using Advanced UI and Suite/Advanced UI Formatted Expressions to Dynamically Configure Command Lines.

- In the Property Manager view: in the Value column for a property. Ensure that you select the Formatted check box so that the formatted expression in the property's value is resolved at run time. The run-time resolution occurs early at run time, before the installation executes the actions in the OnBegin event.

  For more information, see Property Manager View.

- In settings that use string entries, such as the Display Name setting for a feature in the Features view and Text setting for a UI control in the Wizard Interface view.

- In some of the subsettings in which you can enter text to configure exit conditions, detection conditions, eligibility conditions, or other types of conditions—for example, in the Compare To subsetting for the File Comparison, Registry Comparison, and Property Comparison types of condition checks.

  For more information, see Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects.

- In the Events view for an action that sets a property: in the Property Value setting. Ensure that you select Yes in the Format Property Value setting so that the formatted expression in the property's value is resolved at run time.

  For more information, see Working with an Action that Sets a Property in a Suite/Advanced UI Installation.

- In any action that receives an ISuiteExtension pointer: the bstrValue parameter of the FormatProperty method.

  For more information, see:

  - Creating an Extension Condition DLL for an Advanced UI or Suite/Advanced UI Project
  - Working with a DLL File for an Action in a Suite/Advanced UI Installation
  - Working with a Managed-Code Action in a Suite/Advanced UI Installation

- In InstallScript action code: the szValue parameter of the InstallScript function **SuiteFormatString**.

  For more information, see SuiteFormatString.

- In PowerShell actions: the format-suiteproperties cmdlet expands the value of a formatted expression.

  For more information, see Working with a PowerShell Script for an Action in a Suite/Advanced UI Installation.

Writing Object Expressions in Advanced UI and Suite/Advanced UI Projects to Search Target Systems



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

In various settings of Advanced UI and Suite/Advanced UI projects, you can embed object expressions to query target systems for information about a file, a registry entry, or other items. Object expressions use this convention:

```
[@Object(Parameters, ...).Property(Parameters, ...)]
```

Each object expression contains a reference to an object, which is a collection of object-specific properties. Objects and properties may contain parameters.

For example, the following Platform object expression obtains the architecture (x86, x64, IA64, ARM, or Unknown) of the machine on which the Advanced UI or Suite/Advanced UI installation is running:

```
[@Platform.Architecture]
```

The following Package object expression obtains the exit code for the package that has a package GUID of 0F283EC0-E9D1-4D18-801D-0014F6CA96DF once its scheduled operation has completed:

```
[@Parcel(0F283EC0-E9D1-4D18-801D-0014F6CA96DF).ExitCode]
```

Guidelines for Writing Object Expressions

Note the following guidelines when you are writing object expressions:

- Syntax errors, failures of objects in object expressions, and invalid formatted expressions evaluate to an empty string at run time. In some cases, information about the failure to parse and evaluate a formatted expression is logged in the debug log file of the Advanced UI or Suite/Advanced UI installation.
- You can embed object expressions within other formatted expressions, such as within property expressions or other object expressions.

  In the following expression, a Registry object expression is embedded as part of the parameter to a File object expression.

  ```
  [@File([@Registry(HKLM\Software\MyProduct).KeyValue(MyProductPath)]\MyProduct.exe).Version]
  ```

  If the **MyProduct.exe** file is in the location that is specified for the **MyProductPath** value data, the File object expression returns the version of the file. If the file is not found in that location, or if the registry value does not exist, the File object expression returns an empty string.
- You can embed property expressions and other formatted expressions within a parameter of an object expression.

The following File object expression obtains the last modified date and time for a file called **MyProduct.exe**, which is in the 32-bit **ProgramFilesFolder**. (Note that the parameter value of *false* indicates that the installation should not check 64-bit locations.)

```
[@File([ProgramFilesFolder]MyCompany\MyProduct.exe,false).Date(modified)]
```

- The formatted expression parser for Advanced UI and Suite/Advanced UI installations does not require quotation marks. For example, if a parameter in an object expression includes a path or other item that contains one or more spaces, it is not necessary to enclose the path within quotes.
- If you need to include literal square brackets, parentheses, or commas within a parameter of an object expression, embed in the parameter a formatted property expression ([SpecialCharacterString]) or an escaped character expression (A[\,] B[\,] and C) that would resolve to a value that contains the special characters.
- You can use object expressions to perform system searches. You can use the output of one search as the input to a parameter in a different object expression.
- The run-time resolution of object expressions involves two different steps: parsing and evaluation. A parameter in an object expression is parsed first at run time, just as any formatted expression would be parsed. After an entire formatted expression is parsed, the parsed output is then evaluated. No further parsing occurs after the initial parsing is done.

  Thus, if the value of the expression [MYPROP] is [MYPROP], [MYPROP] becomes the parsed output that is evaluated on target systems. This scenario does not result in a recursive lookup.
- For additional sample object expressions, as well as detailed information about each of the available objects, see Object Reference for Expressions in Advanced UI and Suite/Advanced UI Projects.

Minimizing the Number of User Account Control Prompts During Advanced UI and Suite/Advanced UI Installations



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Some of the specific details apply to only the Suite/Advanced UI project type. These differences are noted where appropriate.



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

One of the goals of Windows Vista and later, as well as User Account Control (UAC), is to allow users to run as standard users all of the time. Elevation should rarely be required; if it does occur, it should occur for as short of a duration as possible.

Settings in several different areas of InstallShield affect whether an Advanced UI or Suite/Advanced UI installation triggers UAC consent or credential prompts for elevated privileges. In addition, settings in the packages that are included in an Advanced UI or Suite/Advanced UI installation also affect whether end users see a UAC prompt for elevated privileges. Understanding these different settings will help you create the appropriate UAC experience for your installation when end users run it on Windows Vista and later systems and on Windows Server 2008 and later systems.

UAC-Related Settings in an Advanced UI or Suite/Advanced UI Project

The following settings in an Advanced UI or Suite/Advanced UI project help determine whether UAC prompts are displayed:

- **Required Execution Level**—Use this setting on the Setup.exe tab in the Releases view to specify the minimum execution level required by your installation's **Setup.exe** file. InstallShield uses the value that you select (Administrator, Highest Available, or Invoker) in the application manifest that it embeds in the **Setup.exe** launcher. For more information, see Specifying the Required Execution Level for Your Setup Launcher on Windows Vista and Later Platforms.
- **Require Elevated Privileges**—Use this setting for a package in the Packages view to specify whether the package requires elevated system privileges.
- **Requires Administrative Privileges**—Use this setting for an action that you have added to your project in the Events view to specify whether the action requires administrative privileges.



Project • *Suite/Advanced UI projects include the Events view, as well as support for assigning actions to events.*

UAC-Related Settings in a Package That Is Included in an Advanced UI or Suite/Advanced UI Project

The packages that you include in an Advanced UI or Suite/Advanced UI project have various ways of triggering a UAC prompt for elevated privileges. To learn more, see Minimizing the Number of User Account Control Prompts During Installation.

UAC-Related Run-Time Behavior for an Advanced UI or Suite/Advanced UI Installation

Note the following UAC-related behavior on Windows Vista and later and on Windows Server 2008 and later:

- If Required Execution Level is set to Invoker, none of the actions in the project require administrative privileges, and none of the packages in the installation require administrative privileges, end users should see no UAC prompts during installation. The entire installation (displaying the UI, launching the actions, and running the packages) is run in a nonelevated process.
- If Required Execution Level is set to Invoker, if none of the actions in the project require administrative privileges, and if one or more of the packages in the installation require administrative privileges, end users should see one UAC prompt—plus up to one additional UAC prompt for each reboot—during installation. The UAC prompt is displayed after the end user clicks the button in the UI to install. The packages that have a Require Elevated Privileges value of Yes are launched with elevated privileges in a new separate process. The remaining portions of the UI, the actions, and the packages that have a Require Elevated Privileges value of No are launched in the original nonelevated process.

- If the Required Execution Level is set to Invoker and if one or more of the actions in the project require administrative privileges, end users should see one UAC prompt—plus up to one additional UAC prompt for each reboot—during installation. The timing of the UAC prompt depends on when elevation is first required.

  For example, if the earliest scheduled action that requires administrative privileges is configured to occur during the OnBegin event (which is well before any packages are run), the UAC prompt may be displayed shortly after end users launch the **Setup.exe** file. After end users provide consent or credentials, the installation launches a new elevated process for this action, as well as any subsequent actions and packages that require elevation. Any subsequent actions and packages that do not require elevation are launched in the original nonelevated process, along with the remaining portions of the UI.

  However, if the earliest scheduled action that requires administrative privileges is configured to occur after a package that requires elevated privileges, the UAC prompt is displayed after the end user clicks the button in the UI to install. After end users provide consent or credentials, the installation launches a new elevated process for the package, as well as any subsequent actions and packages that require elevation. Any subsequent actions and packages that do not require elevation are launched in the original nonelevated process, along with the remaining portions of the UI.

Restarting a Target System for an Advanced UI or Suite/Advanced UI Package



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield uses several methods to determine if a target system should be restarted after running a package in an Advanced UI or Suite/Advanced UI installation.

Registry Key Changes for .exe Packages

The target system may need to be restarted if any of the following registry keys have been modified after an .exe package in the Advanced UI or Suite/Advanced UI has been run:

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
- (64-bit) HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx
- (64-bit) HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Session Manager, value of PendingRenameOperations

The registry keys are counted before and after the package is run. If these numbers are not the same, it is assumed that the file is trying to restart the system and exit the installation.

Return Code Specified for a Package in the Advanced UI or Suite/Advanced UI Installation

If the exit code of the package in an Advanced UI or Suite/Advanced UI installation matches the return code that is specified for applicable operation type (install, remove, repair, or modify) for the package in the Reboot Codes setting of the Packages view of the Advanced UI or Suite/Advanced UI project, the target system may need to be restarted.

Return Codes 1641 and 3010

The target system may need to be restarted if a reboot state is returned for an .msi package that is run for a package in an Advanced UI or Suite/Advanced UI installation. The standard Windows Installer reboot return codes are:

- ERROR_SUCCESS_REBOOT_INITIATED (1641)
- ERROR_SUCCESS_REBOOT_REQUIRED (3010)

Behavior Specified for the Package

The Reboot Request setting for a package in the Packages view of the Advanced UI or Suite/Advanced UI project enables you to specify what should happen if any of the aforementioned conditions are or are not applicable to a particular package in the Advanced UI or Suite/Advanced UI project. You can specify different behaviors for each operation type: install, remove, repair, and modify.

For more information, see Specifying the Behavior for an Advanced UI or Suite/Advanced UI Package that Requires a Restart.

Specifying the Behavior for an Advanced UI or Suite/Advanced UI Package that Requires a Restart



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are configuring a new package in an Advanced UI or Suite/Advanced UI project or modifying the settings of an existing one, you can specify how the Advanced UI or Suite/Advanced UI installation should proceed if it appears that the target system needs to be restarted. For example, in some cases, you may want the installation to first prompt the end user before restarting the target system; in other cases, you may want to skip the restart. The

Packages view of the Advanced UI or Suite/Advanced UI project provides the flexibility needed to carry out the response that is appropriate for each type of operation (install, remove, repair, or modify) and for each package in the Advanced UI or Suite/Advanced UI project.



Task ***To specify the behavior for an Advanced UI or Suite/Advanced UI package that requires the target system to be restarted:***

1. Open the Advanced UI or Suite/Advanced UI project that contains the package whose restart behavior you want to configure.
2. In the View List under **Organization**, click **Packages**.
3. In the **Packages** explorer, select the package that you want to configure.
4. Under the appropriate operation (**Install**, **Remove**, **Repair**, or **Modify**), in the **Reboot Request** setting, select the appropriate option.

The following table explains appropriate options for various restart-related behaviors that a package in an Advanced UI or Suite/Advanced UI installation may perform.

Table 5-5 • Restart Behavior for Advanced UI or Suite/Advanced UI Packages

Behavior of the Package in the Advanced UI or Suite/Advanced UI Installation	Guidelines for the Reboot Request Setting
Prompts the end user and, based on the end user's response, may restart the target system	Consider selecting one of the following options for the Reboot Request setting: • **Allow the machine to reboot**—This option is appropriate for most scenarios. • **Always reboot the machine**—Note that this option may conflict with the end user's response to the package's prompt if the end user chooses the option to avoid restarting the target system. If this scenario is possible and you always want to restart the system, consider suppressing the package's prompt, and selecting the appropriate option for the package's Reboot Request setting.
Always triggers a restart; may or may not prompt the end user	Consider selecting one of the following options for the Reboot Request setting: • **Allow the machine to reboot**—This option is appropriate for most scenarios. • **Always reboot the machine**—Note that this option may conflict with the end user's response to the package's prompt if the end user chooses the option to avoid restarting the target system. If this scenario is possible and you always want to restart the system, consider suppressing the package's prompt, and selecting the appropriate option for the package's Reboot Request setting.

Table 5-5 • Restart Behavior for Advanced UI or Suite/Advanced UI Packages (cont.)

Behavior of the Package in the Advanced UI or Suite/Advanced UI Installation	Guidelines for the Reboot Request Setting
Uses a return code or the registry to indicate the need to restart; does not prompt the end user	Consider selecting one of the following options for the Reboot Request setting: • **Prompt, then exit or reboot the machine**—If the target system must be restarted for a subsequent package, you may want to select this option. • **Always prompt, then exit or reboot the machine**—If the target system must be restarted immediately, you may want to select this option. • **Ignore the reboot request**—If the package's restart request is unnecessary, you may want to select this ignore option. • **Delay the prompt, then exit or reboot the machine**—If the restart is not required for a subsequent package, consider selecting this option. If more than one package in the Advanced UI or Suite/Advanced UI installation requires a restart, this option may help minimize the number of restarts. Note that if a restart must occur before a subsequent package runs, and if a restart is not guaranteed to occur before that subsequent package runs, avoid selecting this delay option.
Requires a restart but does not prompt the end user or indicate the need to restart	Consider selecting the following option for the Reboot Request setting: • **Always prompt, then exit or reboot the machine**—If the restart is required, you may want to select this option.

Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Depending on whether an Advanced UI or Suite/Advanced UI installation is running in install mode or maintenance mode, the installation has the ability to check for an updated Advanced UI or Suite/Advanced UI setup launcher that you host on your Web site, and either download it or give end users an option to download it:

- If the installation is running in install mode, it automatically checks for an updated Advanced UI or Suite/Advanced UI setup launcher that you host on your Web site. If an update is available, the installation downloads and launches it.
- If the installation is running in maintenance mode, it checks for an updated Advanced UI or Suite/Advanced UI setup launcher that you host on your Web site. If an update is available, the maintenance wizard page that the installation displays includes an Update button. If the end user clicks that button, the Suite engine downloads it and launches it.

Requirements and Recommendations for Supporting Downloadable Updates

If you want to include support for downloadable updates in your Advanced UI or Suite/Advanced UI installations, the following guidelines are recommended when you are preparing your base setup launchers and your update setup launchers for your Advanced UI or Suite/Advanced UI projects.

- Decide the URL that you will use for hosting a metadata file called **isupdate.xml** file.

  At build time, InstallShield creates this **isupdate.xml** file and places it in an UpdateMetadata subfolder in your project's build folder. This **isupdate.xml** file identifies the different versions of your Advanced UI or Suite/Advanced UI installation and their download locations. You can update this file manually as needed before uploading it to your site.

  You will need to specify an update URL path in your base Advanced UI or Suite/Advanced UI project.
- When you are ready to release an update, upload the **isupdate.xml** file and your update to your site.
- At a minimum, the update setup launcher must be digitally signed. For the best end-user experience, the same digital signature should be used to sign the update setup launcher and the base setup launcher.
- When you are configuring the Advanced UI or Suite/Advanced UI project for the update setup launcher, the run-time location of the packages must be either extracted from the setup launcher or downloaded from the Web. The update setup launcher cannot rely on packages that are stored on the source media.

  Note that your base setup launcher can use packages that are stored on the source media.

  For instructions on how to specify the run-time location for specific packages in an Advanced UI or Suite/Advanced UI project, see Specifying a Run-Time Location for a Specific Package in an Advanced UI or Suite/Advanced UI Project.

  Note that you can override the run-time location for all packages in an Advanced UI or Suite/Advanced UI project. To learn more, see Specifying the Run-Time Location for Advanced UI or Suite/Advanced UI Packages at the Release Level.
- The Advanced UI or the Suite/Advanced UI installation must have an Add or Remove Programs entry. For more information, see Add or Remove Program Entries for an Advanced UI or Suite/Advanced UI Installation.
- The Suite GUID in the update Advanced UI or Suite/Advanced UI project must match that in the base project. This setting is in the General Information view.

Adding Downloadable Update Support



Task ***To add downloadable update support to a base Advanced UI or Suite/Advanced UI project:***

1. Open the Advanced UI or Suite/Advanced UI project for which you want to enable future downloadable updates.
2. In the View List under **Media**, click **Releases**.
3. In the **Releases** explorer, select the release that you want to configure.
4. On the **Setup.exe** tab, in the **Update URL** setting, enter the absolute path URL (starting with either **http://** or **https://**) that you want to use for the future path to the update setup launcher that you will make available for download to target systems.

   The **isupdate.xml** file will need to be uploaded to the same path that you specify in the Update URL setting. For example, if the value of the Update URL setting is **http://MyWebSite.com/path/setup.exe**, the metadata URL would need to be **http://MyWebSite.com/path/isupdate.xml**.

   Note that you can override the location of the setup launcher that you specify in the Update URL setting by specifying the appropriate path in the **isupdate.xml** file. However, the **isupdate.xml** file must be present in the same folder as the .exe file that you specify in the Update URL setting. Otherwise, the downloadable update will not be available to end users.
5. On the **Signing** tab, it is recommended that you enter digital signature information and configure the release to sign the **Setup.exe** launcher.



Task ***To prepare an update setup launcher for an Advanced UI or Suite/Advanced UI project:***

1. Open the Advanced UI or Suite/Advanced UI project that you want to update, and update it as necessary.
2. In the View List under **Media**, click **Releases**.
3. In the **Releases** explorer, select the release that you want to configure.
4. On the **Signing** tab, enter digital signature information and configure the release to sign the **Setup.exe** launcher. It is recommended that you use the same digital signature that was used to sign the base **Setup.exe** launcher.
5. Build the release.

   InstallShield creates the **isupdate.xml** file and places it in an UpdateMetadata subfolder in your project's build folder. The XML file contains an element such as the following to identify different versions and locations of your Advanced UI or Suite/Advanced UI installation:

   ```
   <Update Url="http://www.mysite.com/MyProduct/Setup.exe" Version="1.00.0000" Id="{8E23F36C-5EA8-475D-8EAE-09FA36103975}"/>
   ```

   You can edit the file as needed before uploading it to your site.

   For example, if you want to change the path to the update setup launcher after launch, you can change the value of the Url attribute and then reupload the **isupdate.xml** file to the designated location.

   If you interested in using the same **isupdate.xml** file to define the upgrades for multiple products, you can add the Update element for each of your products to that file.

6. Upload the **isupdate.xml** file and the update setup launcher to your site.

Run-Time Behavior for Downloadable Updates

The run-time behavior for an installation that includes support for downloadable updates varies, depending on the mode in which the installation is running.

Install Mode Behavior

If the base Advanced UI or Suite/Advanced UI setup launcher runs in install mode, the setup launcher checks for the **isupdate.xml** file in the update URL path:

1. If the **isupdate.xml** file is present and it references a newer version of the installation that is available in the specified path, the base Advanced UI or Suite/Advanced UI setup launcher downloads it automatically and then verifies its digital signature.
2. If the **isupdate.xml** file does not include a reference to a newer version of the installation, or if the **isupdate.xml** file is not present, the Advanced UI or Suite/Advanced UI setup launcher checks for a download in the update URL path that was configured in the base project. If the download is available, the base Advanced UI or Suite/Advanced UI setup launcher downloads it automatically and then verifies its digital signature.
3. If a download is not present in the update URL path that was configured in the base project, the base Advanced UI or Suite/Advanced UI setup launcher continues without downloading any updates.

Maintenance Mode Behavior

If the Advanced UI or Suite/Advanced UI setup launcher runs in maintenance mode, the Advanced UI or Suite/Advanced UI setup launcher checks for the **isupdate.xml** file in the update URL path:

1. If the **isupdate.xml** file is present and it references a newer version of the installation that is available in the specified path, the installation sets the **ISUpdateAvailable** property to 1 and displays the MaintenanceUpdateWelcome wizard page—a maintenance wizard page that includes an Update button. In addition, the installation sets the **ISUpdateVersion** property equal to the version of the Suite/Advanced UI or Advanced UI installation that is available for download.

   If the end user clicks the Update button, the setup launcher downloads it and then verifies its digital signature.
2. If the **isupdate.xml** file does not include a reference to a newer version of the installation, or if the **isupdate.xml** file is not present, the installation displays the MaintenanceWelcome wizard page—the standard maintenance wizard page that does not include an Update button.

Digital Signature Verification

If an Advanced UI or Suite/Advanced UI installation downloads an update, it verifies its digital signature before launching it:

1. If the digital signature in the update setup launcher matches that in the base setup launcher, the update setup launcher runs automatically.
2. If the digital signature does not match, or if the base setup launcher is not digitally signed, a security warning is displayed, allowing the end user to choose whether to proceed with the update setup launcher.
3. If the update setup launcher is not digitally signed, the installation fails, and the debug log reports the digital signature problem.

Password-Protecting an Advanced UI or Suite/Advanced UI Installation

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

InstallShield enables you to password-protect the setup launcher for your Advanced UI or Suite/Advanced UI installation. When you password-protect your setup launcher, any end user who wants to run your installation must enter a case-sensitive password.

Task ***To password-protect an Advanced UI or Suite/Advanced UI installation:***

1. Add password protection for your setup launcher:
   a. In the View List under **Media**, click **Releases**.
   b. In the **Releases** explorer, select the release that you want to configure.
   c. Click the **Setup.exe** tab. The setup launcher's settings are displayed.
   d. In the **Password Protect Launcher** setting, select **Yes**.
   e. In the **Launcher Password** setting, specify the case-sensitive password that you want require end users to specify.
2. Optionally add a predefined wizard page where end users can enter the password:
   a. In the View List under **User Interface**, click **Wizard Interface**.
   b. In the **Wizard Interface** explorer, right-click **Wizard Pages** and then click **Add Predefined Page**. The **User Interface Wizard** opens.
   c. In the **Task page** list, select **Enter installation's password**.
   d. In the **Subsequent page** list, indicate where in the wizard page sequence to schedule the password wizard page. The page that you select in this list will be displayed immediately after the new password wizard page at run time.

Note *• When you add the predefined password wizard page to your project, InstallShield adds a placeholder image control to it so that you can add a lock image to the page. Ensure that you either configure the control's Resource setting to indicate the file that you want to display with this control, or, to exclude an image, delete the image control.*

By default, the predefined password wizard page uses the **ISPassword** property to store the password value that the end user enters. The Advanced UI or Suite/Advanced UI installation compares the value of the **ISPassword** property with the password that is stored in the password in the **Setup.xml** file. The installation advances to the next wizard page if the password that the end user entered and the one that was configured in the project match.

For silent Advanced UI and Suite/Advanced UI installations, end users can use the `/password` command-line parameter to specify the password from the command line. To learn more, see Advanced UI and Suite/Advanced UI Setup.exe Command-Line Parameters.

Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The following information provides background information to help you resolve issues with Advanced UI and Suite/Advanced UI installations.

Reasons Why an Advanced UI or Suite/Advanced UI Installation May Run in Maintenance Mode

An Advanced UI or Suite/Advanced UI installation may run in maintenance mode in the following scenarios:

- An end user chooses to modify the product through the product's Add or Remove Programs entry.
- An end user reruns the Advanced UI or Suite/Advanced UI installation.
- The detection conditions for one or more of the primary packages in the Advanced UI or Suite/Advanced UI installation are not configured properly. For information on configuring conditions for an Advanced UI or Suite/Advanced UI package, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.
- A package in the Advanced UI or Suite/Advanced UI installation is configured erroneously to be a primary package instead of a dependency package, and the package is already installed on the target system.

  InstallShield creates maintenance mode conditions automatically, based on the detection conditions of all of the primary packages in the Advanced UI or Suite/Advanced UI project. The Advanced UI or Suite/Advanced UI engine uses these conditions to determine whether the Advanced UI or Suite/Advanced UI installation runs in maintenance mode. If any of the primary packages' detection conditions evaluate as true (that is, if one or more of the primary packages seem to be installed already), the maintenance mode condition is true, and the Advanced UI or Suite/Advanced UI installation runs in maintenance mode.

  Use the Package Type setting in the Packages view to indicate whether a package is a primary package or a dependency package.

To learn more about mode conditions, see Triggering Install Mode or Maintenance Mode of an Advanced UI or Suite/Advanced UI Installation.

Logging an Advanced UI or Suite/Advanced UI Installation to Troubleshoot Issues

If you are troubleshooting issues with an Advanced UI or Suite/Advanced UI installation, start with a debug log file. Debug log files provide some insight into the operations that an Advanced UI or Suite/Advanced UI installation performs at run time. To create a debug log file, run the Advanced UI or Suite/Advanced UI installation from the command line with `/debuglog` parameter. To generate a log file named **InstallShield.log** in the same directory as the **Setup.exe** file, pass just the command-line parameter. Note that this does not work if the **Setup.exe** file is in a read-only location. For example:

```
Setup.exe /debuglog
```

To specify the name and location of the log file, pass the path and name, as in the following example:

```
Setup.exe /debuglog"C:\PathToLog\setupexe.log"
```

Advanced UI and Suite/Advanced UI debug log files contain the following information:

- Initialization information from the Advanced UI or Suite/Advanced UI engine
- Property values and property value changes
- Information about the actions and installed states of features
- Information about the detected state and eligibility of packages
- Package command lines and launch result information
- For an InstallScript package—Information that is logged by the InstallScript function SuiteLogInfo

The default Advanced UI or Suite/Advanced UI user interface in Advanced UI and Suite/Advanced UI projects also logs information to the debug log file; however, in general, this information can be ignored. (UI information is typically prefixed with `UI DLL:` in the log.)

Note that when the Advanced UI or Suite/Advanced UI engine writes feature, package state, and Advanced UI or Suite/Advanced UI mode information to the log file, it writes only numeric values for the states. The following tables map the numeric values to state representations.

Table 5-6 • Feature Action States

Log Value	State
0	No action will be taken for the feature.
1	The feature will be installed.
2	The feature will be removed.

Table 5-7 • Feature Installed States

Log Value	State
0	The feature is not currently installed.
3	The feature is installed.

Table 5-8 • Package States

Log Value	Installed State	Action State
0	The package is not installed.	—
1	The package is currently installed.	The package will run its install operation.
2	—	The package will run its modify operation.
3	—	The package will run its repair operation.
4	—	The package will run its remove operation.
5	—	No action will be performed for the package.

Table 5-9 • Advanced UI and Suite/Advanced UI Installation Modes

Log Value	Install Mode
0	First-time installation
1	Maintenance/UI maintenance mode selection
2	Maintenance/modify
3	Maintenance/remove
4	Maintenance/repair
5	Stage only (The installation is run in stage-only mode, in which the Advanced UI or Suite/Advanced UI installation is staged—or uncompressed and copied to a specified location. Note that none of the packages in the installation are run in this mode.)

Example Debugging Walkthrough

Debug log files can be useful in any number of scenarios where an Advanced UI or Suite/Advanced UI installation is not behaving in an expected manner. For example, suppose a Suite/Advanced UI project contains two packages: one .exe file and one .msi file, each associated with their own feature in the Suite/Advanced UI installation. At run time on some machines, even though both features appear to be selected, only the .exe package is installing. To begin troubleshooting this behavior, generate a debug log file on one of the machines that is encountering this behavior. Once the log file is available, verify that both features were in fact selected to be installed. The Suite/Advanced UI records logs information such as the following about each feature:

```
9-21-2011[03:07:04 PM]: Engine: determining suite feature states

9-21-2011[03:07:04 PM]: Initializing state for feature 'NewFeature'
```

```
9-21-2011[03:07:04 PM]: Default action state 1 for mode 0

9-21-2011[03:07:04 PM]: Initial feature state: 1

9-21-2011[03:07:04 PM]: Final feature state: 1

…

9-21-2011[03:07:04 PM]: Initializing state for feature 'NewFeature1'

9-21-2011[03:07:04 PM]: Default action state 1 for mode 0

9-21-2011[03:07:04 PM]: Initial feature state: 1

9-21-2011[03:07:04 PM]: Final feature state: 1
```

The above information indicates the initial states of features NewFeature and NewFeature1. Both have been initialized to a state of 1, indicating they are selected for installation. So the features in this Suite/Advanced UI installation do not appear to be an issue. Continuing on in the log, the following information is provided:

```
9-21-2011[03:07:10 PM]: Engine: setting parcel states as determined by feature selections

9-21-2011[03:07:10 PM]: Feature NewFeature setting parcel states, parent override: no, override state: 0

9-21-2011[03:07:10 PM]: Requesting action state 1 for parcel '{BA3EAE7A-0701-4CE0-9224-4F5C0F135792}'

9-21-2011[03:07:10 PM]: Containing feature is request state change to parcel {BA3EAE7A-0701-4CE0-9224-
    4F5C0F135792}, feature request: 1
```

At this point in the Suite/Advanced UI installation, the UI DLL has completed the UI selection phase and gathered all necessary data for the installation to start. The Suite/Advanced UI engine now determines what packages (referred to as *parcels* in the log file) will be installed based on the features with which they are associated. This information indicates package {BA3EAE7A-0701-4CE0-9224-4F5C0F135792} (this GUID comes from the Package GUID setting that is set in each package in an Advanced UI or Suite/Advanced UI project) is being requested to install (action state 1) from feature NewFeature (whose action state is selected to install). Looking forward a couple lines in the log provides the following information:

```
9-21-2011[03:07:10 PM]: Parcel is ineligible or the current parcel state and install mode to not allow
    parcel configuration
```

That line indicates that the package (the .msi package that is not installing as expected) has become ineligible. Looking for this package with the given package ID in the Packages view in the Suite/Advanced UI project shows that the package contains an eligibility condition. The condition indicates that the package is eligible if either (a) a package with a particular .msi product code and package code are already installed, or (b) an .msi package with a specific product code is installed, and the product version is not greater than the .msi package's product version. The purpose of this condition is to prevent the package from installing if a newer version of this product is already installed.

Investigating further on the machine that produced this behavior shows that a newer version of the product is already present on the machine, causing the .msi package in the Suite/Advanced UI installation to become ineligible.

Debugging InstallScript Actions in a Suite/Advanced UI Installation

To troubleshoot issues with InstallScript actions in a Suite/Advanced UI installation, you can use the InstallScript Debugger. To debug the InstallScript actions in a Suite/Advanced UI installation, run the Suite/Advanced UI installation from the command line with `/d` parameter.

Note *• Debugging InstallScript code requires the debug-information file* **Setup.dbg** *to be available. In addition, in order to debug an installation on a system other than your development machine, you need to copy certain files from your development machine to the debugging machine. To learn more, see Debugging an Installation on Any Computer.*

The following command runs your InstallScript custom actions in the InstallScript Debugger:

```
Setup.exe /d
```

If the installation is located in a path at which they were originally compiled, you also need to specify the path to your debug symbols file (.dbg):

```
Setup.exe /d"Path-to-Setup.dbg"
```

Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are configuring the settings for a package in an Advanced UI or Suite/Advanced UI project, you can specify whether you want the package to generate a log file if the Advanced UI or Suite/Advanced UI installation is launched from the command line with the `/log` command-line parameter. Depending on the type of package, you can also configure one or two other settings to specify information such as which log options you want the Advanced UI or Suite/Advanced UI installation to pass to the package when the log file is being created.

Note that if a package does not have support for logging, it is recommended that you leave logging support disabled for that package.

Enabling Package Logging Support

Task ***To enable logging support for a package in an Advanced UI or Suite/Advanced UI installation:***

1. Open the Advanced UI or Suite/Advanced UI project that contains the package for which you want to enable logging.
2. In the View List under **Organization**, click **Packages**.
3. In the **Packages** explorer, select the package that you want to configure.

4. On the **Common** tab, in the **Enable Logging Support** setting, select **Yes**.
5. For an .msi or .msp package, configure the following optional subsettings as needed:

   - **Log File**—Specify a name for the log file. Do not include a path for the file; the Advanced UI or Suite/Advanced UI `/log` command-line parameter lets end users specify the directory for the package log files.

     If you leave this setting blank, the name of the log file that the installation creates is *PackageGUID*.**log**, where *PackageGUID* is the GUID that is assigned to the package in the Package GUID setting in the Packages view.

   - **Log Options**—Specify the Windows Installer log `/L` flags that you want the package to use when generating the log file. For example, to generate a log file that logs everything verbosely, enter the following in this setting:

     *v

     For additional valid flags, see the /L description.

     If you leave this setting blank, the asterisk (*) and v flags are used to generate the log file.

   For an .exe package, configure the following subsetting:

   - **Logging Command Line**—Specify the command line that the Advanced UI or Suite/Advanced UI installation should pass to the .exe package to enable the logging. Include any appropriate supported flags. If the .exe package that you are configuring supports it, include a path that references the Advanced UI or Suite/Advanced UI property **ISLogDir**, enclosed within square brackets, in place of the path to the directory where the log file should be created.

     For example, to generate a log file that logs everything verbosely for an .msi package that is run by a **Setup.exe** file that InstallShield built, enter the following command line:

     ```
     /v"/l*v \"[ISLogDir]FileName.log\""
     ```

     The Advanced UI or Suite/Advanced UI installation replaces **[ISLogDir]** with the path to the folder that will contain the log file. The Advanced UI or Suite/Advanced UI `/log` command-line parameter lets end users specify the path to the directory for the package log files.

Generating Package Log Files While Running an Advanced UI or Suite/Advanced UI Installation

To create a package log file, run the Advanced UI or Suite/Advanced UI installation from the command line with the `/log` parameter. The `/log` parameter lets you generate a log file for each package in the Advanced UI or Suite/Advanced UI installation for which logging is enabled.

To generate package log files in a particular folder, pass the folder path with the `/log` parameter to the Advanced UI or Suite/Advanced UI **Setup.exe** file. You can optionally separate the `/log` parameter and the path with a colon. For example, both of the following command lines generate a log file in the **PathToLogFiles** folder:

```
Setup.exe /log"C:\PathToLogFiles"
```

```
Setup.exe /log:"C:\PathToLogFiles"
```

Note that the path to the log file location must already exist.

To generate package log files in the *%TEMP%* directory, leave out the path. For example:

```
Setup.exe /log
```

The Advanced UI or Suite/Advanced UI property **ISLogDir** stores the path to the directory that contains the package log files.



Tip *• You can also use the logging system policy or the MsiLogging property to generate log files for Windows Installer–based packages in an Advanced UI or Suite/Advanced UI installation.*

6

Designing InstallShield Prerequisites and Other Redistributables

InstallShield provides many third-party redistributables that you can incorporate into projects as InstallShield prerequisites, merge modules, and InstallScript objects. You can also create your own redistributables and distribute them to other installation developers. The following sections describe each of these types of redistributables.

InstallShield Prerequisites



Project *• The following project types include support for InstallShield prerequisites:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

InstallShield also includes support for including InstallShield prerequisites as packages in Advanced UI and Suite/Advanced UI projects. For more information, see Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project.

An InstallShield prerequisite is an installation for a product or technology framework that is required by your product. Some examples of InstallShield prerequisites that are included with InstallShield are Java Runtime Environment (JRE) and SQL Server Express Edition. You can add any of the existing InstallShield prerequisites to your installation projects and configure many of their settings. You can also create your own InstallShield prerequisites, and add them to your projects.



Project *• In Windows Installer–based installations, including InstallShield prerequisites in your project enables you to chain multiple installations together, bypassing the Windows Installer limitation that permits only one Execute sequence to be run at a time. The* **Setup.exe** *setup launcher serves as a bootstrap application that manages the chaining.*



Note *• Unlike Windows Installer 4.5 chaining, the InstallShield prerequisite installations are not processed as a single transaction; that is, successful installations are not rolled back after failures in later prerequisites. To learn more about*

Windows Installer 4.5 chaining support, see Configuring Multiple Packages for Installation Using Transaction Processing.

For information on configuring the settings for the InstallShield prerequisites that are available in InstallShield, as well as details on how to create your own InstallShield prerequisites, see Defining InstallShield Prerequisites.

Types of InstallShield Prerequisites

When you add an InstallShield prerequisite to a Basic MSI, InstallScript, or InstallScript MSI project, it is considered to be the *setup prerequisite* type of InstallShield prerequisite by default. That is, it is treated as a base application or component that must be installed on the target machine before your product can be installed. The installation for a setup prerequisite runs before the main installation runs if the prerequisite is not already installed on the system.

Basic MSI projects enable you to associate InstallShield prerequisites with features in your main installation. When an InstallShield prerequisite is associated with one or more features, it is called a *feature prerequisite*; it is installed if one or more features that contain the prerequisite are installed at run time and if the prerequisite is not already installed on the system. Thus, if a feature has a condition that is not met on the target system, or if the end user chooses not to install the feature, the feature is not installed. As a result, none of its associated feature prerequisites are installed, unless the feature prerequisites are also associated with other features that are installed.

To learn more about these two types of InstallShield prerequisites, see Setup Prerequisites vs. Feature Prerequisites.

Merge Modules

InstallShield enables you to create your own merge modules that can be used in any of your installation projects or distributed for use by other installation developers.

To learn more, see Designing Merge Modules.



***Note** • Many of the merge modules included in the Redistributables view are authored by Microsoft or another third party. InstallShield distributes these modules as a courtesy to assist you in creating your installation project. However, InstallShield cannot modify or fix any problems that may exist within third party-authored modules. You are encouraged to contact the vendor regarding issues with specific third party-authored modules.*

InstallScript Objects

InstallShield enables you to create your own InstallScript objects that can be used in any of your installation projects or distributed for use by other installation developers.

To learn more, see Creating InstallScript Objects.

Defining InstallShield Prerequisites

Project • *The following project types include support for InstallShield prerequisites:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

All of these project types include support for the setup prerequisite type of InstallShield prerequisite. Basic MSI projects include support for the feature prerequisite type.

InstallShield also includes support for including InstallShield prerequisites as packages in Advanced UI and Suite/Advanced UI projects. For more information, see Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project.

Use the InstallShield Prerequisite Editor in InstallShield to modify the settings for any of the prerequisites that are included with InstallShield. You can also create custom prerequisites using this tool and then add them to your projects.

Specifying your own InstallShield prerequisites and modifying existing ones enables you to set options such as the following:

- The URL from where the prerequisite files should be downloaded to the target machine
- Conditions under which the prerequisite should be installed
- Other prerequisites on which a particular prerequisite is dependent
- The command line for the prerequisite
- The command line to be used if the installation is running in silent mode
- Whether the target machine should be restarted after installation of a prerequisite
- Whether the administrative privileges are required to install the prerequisite
- Whether the installation should display a message box that enables end users to choose whether to install the prerequisite

Creating an InstallShield Prerequisite

Task ***To create a new InstallShield prerequisite:***

1. On the **Tools** menu, click **Prerequisite Editor**. The **InstallShield Prerequisite Editor** opens.
2. Configure the settings on each of the tabs as needed.
3. On the **File** menu, click **Save**.

Modifying an Existing InstallShield Prerequisite

Task ***To modify an existing InstallShield prerequisite:***

1. On the **Tools** menu, click **Prerequisite Editor**. The **InstallShield Prerequisite Editor** opens.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.
3. Select the InstallShield prerequisite file (.prq) by browsing, and then click **Open**.
4. Make changes as needed.
5. On the **File** menu, click **Save**.

Setting Properties for an InstallShield Prerequisite

The Properties tab of the InstallShield Prerequisite Editor is where you specify a description and a unique identifier for an InstallShield prerequisite.

Task ***To set properties for an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Properties** tab.
3. In the **Unique identifier for InstallShield prerequisite** box, type a unique file identifier for the prerequisite or leave the default value as is. This could be the name of the prerequisite application or component, or a GUID.

   Note • *Every time you open the InstallShield Prerequisite Editor to create a new prerequisite, a new GUID is generated and listed in the* ***Unique identifier for InstallShield prerequisite*** *box.*
4. In the **Description** box, type an overview of the InstallShield prerequisite. This description is displayed under the **Overview** heading when you select an InstallShield prerequisite in the **Redistributables** view.

Specifying an Alternate URL for a .prq File

You may want to specify an alternate URL for your .prq file if both of the following conditions are true:

- You have specified that one or more InstallShield prerequisites in your installation should be downloaded from the Web instead of being included in the **Setup.exe** file or on the source media. For more information, see Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level.
- It is possible that later you might want to redirect your end users to a different URL to download an InstallShield prerequisite's application files.

In this scenario, the .prq file that you include with your **Setup.exe** installation has an alternate URL specified. When an end user runs the installation, the target machine visits the alternate URL specified in the **Setup.exe** .prq file, downloads that .prq file, and uses the file URLs specified in the alternate .prq file to download the necessary files for the prerequisite application.

Task ***To specify an alternate URL for a .prq file:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Properties** tab.
3. In the **Alternate location to download .prq from if prerequisite files are being downloaded** box, type the alternate URL for your .prq file. For example:

   http://www.mywebsite.com/MyPrq.prq

Note • *InstallShield prerequisites do not have support for FTP URLs.*

Setting Installation Conditions for an InstallShield Prerequisite

You need to specify installation conditions that determine whether an InstallShield prerequisite is already installed on the target machine. Failure to do so causes problems because the target system behaves as if the prerequisite was not properly installed. You can also create installation conditions that specify operating system, registry, or file requirements. The Conditions tab of the InstallShield Prerequisite Editor is where you set installation conditions.

If the InstallShield prerequisite should be installed on some—but not all—operating systems, you can create multiple operating system conditions for the prerequisite. If a target machine meets any one individual operating system condition, it meets the operating system requirements for that prerequisite.

If a condition evaluates as true at run time, that condition is met. The InstallShield prerequisite is installed on the target machine if the following are true:

- The target machine meets any of the operating system conditions and all of the other conditions that are listed on the Conditions tab.
- For feature prerequisites only (that is, an InstallShield prerequisite that is associated with one or more features in the main installation)—The feature that contains the feature prerequisite must be installed. Thus, if the feature has a condition that is not met on the target system, or if the end user chooses not to install the feature, the feature is not installed. As a result, none of its associated feature prerequisites are installed, unless the feature prerequisites are also associated with other features that are installed.

Note • *Because installation of prerequisites such as Windows service packs is not supported, the operating system version information is not expected to change. Therefore, the operating system conditions are not considered in verifying whether a prerequisite was installed correctly.*

Adding a New Installation Condition for an InstallShield Prerequisite



Task ***To add a new installation condition for an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Conditions** tab.
3. Click **Add**. The **Prerequisite Condition** dialog box opens.
4. Select the appropriate option for the type of condition and then set the properties for the condition as needed.

The InstallShield Prerequisite Editor adds the condition to the Conditions tab.

If a condition evaluates as true at run time, that condition is met. The InstallShield prerequisite is installed on the target machine if the following are true:

- The target machine meets any of the operating system conditions and all of the other conditions that are listed on the Conditions tab.
- For feature prerequisites only (that is, an InstallShield prerequisite that is associated with one or more features in the main installation)—The feature that contains the feature prerequisite must be installed. Thus, if the feature has a condition that is not met on the target system, or if the end user chooses not to install the feature, the feature is not installed. As a result, none of its associated feature prerequisites are installed, unless the feature prerequisites are also associated with other features that are installed.

For example, you might have an InstallShield prerequisite that has several different conditions:

- A specific file is not present on the target machine. In this example, if this file is missing, it indicates that an application that your product requires must be installed; otherwise, your product should not be installed.
- The target machine has Windows XP.
- The target machine has Windows Vista.

In this example, if the target machine is running Windows XP or Windows Vista and the InstallShield prerequisite is not associated with a feature, the prerequisite is installed, but only if the specified file is not present. The same is true if the InstallShield prerequisite is associated with a feature that the installation installs at run time.



Note *• Because installation of prerequisites such as Windows service packs is not supported, the operating system version information is not expected to change. Therefore, the operating system conditions are not considered in verifying whether a prerequisite was installed correctly.*

Adding an Operating System Condition for an InstallShield Prerequisite

If the InstallShield prerequisite should be installed on some—but not all—operating systems, you can create multiple operating system conditions for the prerequisite. If a target machine meets any one individual operating system condition, it meets the operating system requirements for that prerequisite.

Task ***To add a new operating system requirement for an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Conditions** tab.
3. Click the **Add** button. The **Prerequisite Condition** dialog box opens.
4. Select the Setup is running on a specified platform option.
5. Do one of the following:
   - In the **Select which platform the prerequisite should be run on** box, select the operating system that is required on the target system for the prerequisite.
   - In the **Select which platform the prerequisite should be run on** box, select **Custom**. Then configure the operating system settings as appropriate.

   Tip • *You can also use the values from an existing predefined operating system condition as the basis for a new custom condition. To do so: In the* ***Select which platform the prerequisite should be run on*** *box, select the appropriate predefined operating system, and then change the setting to* ***Custom****. Then you can further configure the operating system condition by specifying additional values in the* ***Custom*** *area.*
6. To specify a range of service pack numbers for which this prerequisite condition is true, use the **Service Packs** boxes. For example, if target system must have service pack 2, 3, or 4, enter 2 in the first box and 4 in the second box. To target all future service packs, leave the second box blank. To target only service pack 3, enter 3 in both boxes.
7. Click **OK**.

The InstallShield Prerequisite Editor adds the operating system condition to the Conditions tab.

Note • *Instead of storing in the .prq file the value that you selected in the* ***Select which platform the prerequisite should be run on*** *box, the InstallShield Prerequisite Editor stores the underlying operating system settings that are associated with the value that you selected. Therefore, if you select the Custom option but do not change or configure any settings in the Custom area, the InstallShield Prerequisite Editor does not set the value of this box to Custom the next time that you reopen the Prerequisite Condition dialog box for the selected condition. Instead, it sets the value of the box to the predefined operating system option.*

Modifying an Installation Condition for an InstallShield Prerequisite

Task ***To modify an existing condition:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Conditions** tab.
3. Select the condition that you would like to modify.

4. Click the **Modify** button. The **Prerequisite Condition** dialog box opens.
5. Modify the condition as needed.

The InstallShield Prerequisite Editor updates the modified condition on the Conditions tab.

Removing an Installation Condition from an InstallShield Prerequisite



Task ***Remove a condition from an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Conditions** tab.
3. Select the condition that you would like to remove.
4. Click **Remove**.

Specifying Files for an InstallShield Prerequisite

When you are creating a new InstallShield prerequisite, you must specify the installation files that should be included with the prerequisite. You can also modify the list of files for an existing InstallShield prerequisite. The Files to Include tab of the InstallShield Prerequisite Editor is where you specify the files.



Task ***To add one file for an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Files to Include** tab.
3. Click **Add**. The **New File** dialog box opens.
4. In the **File** box, type the path and name of the file. To find the file by browsing, click the ellipsis (**...**) button.
5. If end users should be able to download the file from the Web, specify the URL in the **URL to file** box. This URL is the same one that InstallShield uses when installation authors use the **Redistributables** view to download an InstallShield prerequisite from the Internet to their local machines.

   For example, if the URL for the file is **http://www.mywebsite.com/Folder1/Notepad.exe**, enter the following in this box:

   http://www.mywebsite.com/Folder1



Note • *InstallShield prerequisites do not have support for FTP URLs.*

Task ***To add more than one file at a time for an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, click the **Files to Include** tab.
2. Click **Add Multiple Files**. The **Open** dialog box opens.
3. Select the files that you would like to add. To select multiple consecutive files, select the first file, press and hold SHIFT, and select the last file. To select multiple nonconsecutive files, select the first file, press and hold CTRL, and select each additional file.
4. Click **Open**.

Note • *Adding files by using this method does not also set the URL where those files can be downloaded. To set the URL for multiple files, see* *Specifying URLs for Downloading InstallShield Prerequisites*.

Task ***To modify the settings for an existing file:***

1. In the **InstallShield Prerequisite Editor**, click the **Files to Include** tab.
2. Select the file whose name, path, or URL you would like to modify.
3. Click **Modify**. The **New File** dialog box opens.
4. Modify the settings as needed.

Task ***Remove a file from an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, click the **Files to Include** tab.
2. Select the file that you would like to remove.
3. Click **Remove**.

Specifying URLs for Downloading InstallShield Prerequisites

If end users should be able to download the files for an InstallShield prerequisite in your installation, you must specify the URL on the Files to Include tab of the InstallShield Prerequisite Editor.

Note • *InstallShield prerequisites do not have support for FTP URLs.*

Task ***To specify the URL of one or more files for an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Files to Include** tab.
3. Select one or more files for which you would like to specify the URL. To select multiple consecutive files, select the first file, press and hold SHIFT, and select the last file. To select multiple nonconsecutive files, select the first file, press and hold CTRL, and select each additional file.
4. Click **Set File(s) URL**. The **Set File(s) URL** dialog box opens.
5. In the **URL to File's Parent Folder** box, type the URL for the parent directory of the selected file or files. This URL is the same one that InstallShield uses when installation authors use the **Redistributables** view to download an InstallShield prerequisite from the Internet to their local machines.

   For example, if you selected files called **Notepad.exe** and **Paint.exe**, and the URLs for these files are **http://www.mywebsite.com/Folder1/Notepad.exe** and **http://www.mywebsite.com/Folder1/Paint.exe**, enter the following in this box:

   http://www.mywebsite.com/Folder1
6. Click **OK**.

Tip *• As an alternative to the above steps, you can also specify the URL for an individual file when you add it to the list of prerequisite files on the Files to Include tab. For more information, see Specifying Files for an InstallShield Prerequisite.*

Specifying Parameters for Installing an InstallShield Prerequisite

The Application to Run tab of the InstallShield Prerequisite Editor is where you specify how a prerequisite should be installed on the target machine.

Task ***To specify parameters for installing an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Application to Run** tab.
3. In the **Specify the application you wish to launch** list, select the file—typically a **Setup.exe** setup launcher or an .msi file—that should be launched on the target machine if the InstallShield prerequisite is installed. Only files that have been specified on the **Files to Include** tab are included in this list.

   If the main installation is a Basic MSI or InstallScript MSI installation, and if you specify an .msi file and you indicate on the Behavior tab that the progress should be shown, the **Setup.exe** setup launcher captures progress messages and uses Windows Installer APIs instead of **MsiExec.exe** to launch the .msi package at run time.

If you specify any other file type, or if you specify an .msi file for which progress should not be shown, the **Setup.exe** setup launcher runs the file with either the open verb (for .msi and .exe files) or the default verb (for all other file types) at run time.



Project *• If you specify an .msi file and you indicate on the Behavior tab that the progress should be shown, the* **Setup.exe** *setup launcher captures progress messages and uses Windows Installer APIs instead of* **MsiExec.exe** *to launch the .msi package at run time.*

If you specify any other file type, if you specify an .msi file for which progress should not be shown, or if the main installation is an InstallScript installation, the **Setup.exe** setup launcher runs the file with either the open verb (for .msi and .exe files) or the default verb (for all other file types) at run time.

4. If the Windows Installer engine, the .NET Framework, or both must be installed before this InstallShield prerequisite is installed, select the **Requires Windows Installer engine and/or .NET Framework to be installed first** check box.



Note *• Selecting this check box does not add the Windows Installer engine or the .NET Framework to your installation. It only specifies that if they are included in the installation, they should be installed before this InstallShield prerequisite is installed. To include the Windows Installer engine or the .NET Framework with your installation, you must add them to your project. To learn how, see Adding Windows Installer Redistributables to Projects or Adding .NET Framework Redistributables to Projects.*

5. If applicable, in the **Specify the command line for the application** box, type the command line for the file selected in the **Specify the application you wish to launch** list. Do not include the name of the file in this box.

   For more details, see Specifying Command-Line Parameters for an InstallShield Prerequisite.

6. If applicable, in the **Specify the command line for the application when the setup is running in silent mode** box, type the appropriate command line. Do not include the name of the file in this box.

   For more details, see Specifying Command-Line Parameters for an InstallShield Prerequisite.

7. If the selected InstallShield prerequisite application requires that the target machine be restarted after the application is installed, type the return code in the **Specify the return code (in decimal) the application returns if a reboot is required** box.



Tip *• If multiple return codes exist, list each one separated by a comma.*

- *If you do not know the return codes for the file that you are launching as the InstallShield prerequisite, contact the author of the file.*
- *For more information on InstallShield prerequisites that require a restart, see Restarting a Target Machine for InstallShield Prerequisite Installations.*

Specifying Command-Line Parameters for an InstallShield Prerequisite

The Application to Run tab of the InstallShield Prerequisite Editor lets you specify command-line parameters that you want to be used when the InstallShield prerequisite is launched on the target system.



Task ***To specify command-line parameters that you want to be used when an InstallShield prerequisite installation is launched:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Application to Run** tab.
3. In the **Specify the command line for the application** box, type any valid command-line parameters for the file that is selected in the **Specify the application you wish to launch** list. Do not include the name of the file in this box.
4. If you want to specify the command line that should be used for the InstallShield prerequisite if the main installation is running in silent mode, type any valid command-line parameters in the **Specify the command line for the application when the setup is running in silent mode** box. Do not include the name of the file in this box.

   The command-line parameters that you enter in this setting are also used if the prerequisite is configured to be hidden—that is, if the **The prerequisite should be hidden from the installation list** check box on the Behavior tab is selected.



Note • *Using the* `/s` *command-line parameter to launch an installation that includes an InstallShield prerequisite does not automatically run the prerequisite installation silently. You may also need to specify a valid silent command-line parameter for the InstallShield prerequisite in the* ***Specify the command line for the application when the setup is running in silent mode*** *setting on the Application to Run tab.*

Also note that no command-line parameters are passed to the InstallShield prerequisite if you leave the ***Specify the command line for the application when the setup is running in silent mode*** *box blank and one or both of the following conditions are also true:*

- *The main installation is run silently.*
- *The* ***The prerequisite should be hidden from the installation list*** *check box on the Behavior tab is selected for the prerequisite.*

Therefore, if you specify standard command-line parameters, it is recommended that you also specify silent command-line parameters.

How the Setup Launcher Passes the Command-Line Parameters to the InstallShield Prerequisite

If the InstallShield prerequisite installation is an .msi package and you indicate on the Behavior tab that the progress should be shown, the setup launcher uses Windows Installer APIs—instead of MsiExec.exe—to launch the .msi package with the command-line parameters that you specify on the Application to Run tab. Under these circumstances, you can enter any of the following types of parameters for the command line:

- Windows Installer property names and values in the following format:

  ```
  PROPERTY=VALUE
  ```

- `/L` (as well as any variants such as `/L*v`)
- `/log`
- `/q` (as well as any variants such as `/qb+!`)

- `/quiet`

Note that if you specify that the progress should be shown, the user interface of the prerequisite's .msi package is not displayed by default. If you want to override this behavior, you can specify `/qf` as a command-line parameter.

If the InstallShield prerequisite installation is any other file type, or it is an .msi package for which progress should not be shown, the setup launcher runs the file with either the open verb (for .msi and .exe files) or the default verb (for all other file types) at run time. Therefore, in the case of an .msi package for which progress should not be shown, you can use any of the command-line parameters that **MsiExec.exe** accepts. For a list of the supported command-line parameters, see Command-Line Options and Standard Installer Command-Line Options in the Windows Installer Help Library.

If your InstallShield prerequisite installation is a **Setup.exe** file that was created with InstallShield, the formatting that is done for that prerequisite command-line parameters that you specify on the Application to Run tab differs depending on whether you associated the InstallShield prerequisite with a feature in your project. If the InstallShield prerequisite is not associated with a feature, it is called a *setup prerequisite*. If the InstallShield prerequisite is associated with one or more features, it is called a *feature prerequisite*.



***Note** • To learn more about the differences between setup prerequisites and feature prerequisites, see Setup Prerequisites vs. Feature Prerequisites.*

In the setup prerequisite case, the setup launcher supports property substitution through the command-line parameters for the following properties:

- **SETUPEXEDIR**—This property identifies the path to the setup launcher file. For example, if the path to the setup launcher is **C:\MySetups\MyApp\Setup.exe**, the value of **[SETUPEXEDIR]** is **C:\MySetups\MyApp**.
- **SETUPEXENAME**—This property identifies the name of the setup launcher file that was created when the project was built. The installation updates the value of this property at run time if the setup launcher file was renamed. The following path identifies the full path to this file:

  [SETUPEXEDIR]\[SETUPEXENAME]
- **ISPREREQDIR**—This property identifies the path to the running InstallShield prerequisite. The path includes the final slash. You can use this property to refer to files within the InstallShield prerequisite—for example, **[ISPREREQDIR]MyConfigFile.ini**.
- **ProductLanguage**—This property identifies the language that the installation should use for any strings in the user interface that are not authored into the database.

For a feature prerequisite, the command-line parameters that you specify on the Application to Launch tab are formatted at run time like a formatted text field. Therefore, you can use the aforementioned properties at the command line, as well as any of the standard command-line parameters that **Setup.exe** supports. For a list of supported command-line parameters, see Setup.exe and Update.exe Command-Line Parameters.

Restarting a Target Machine for InstallShield Prerequisite Installations

InstallShield uses several methods to determine if a target machine should be restarted after running the InstallShield prerequisite.

Registry Key Changes

The target machine may need to be restarted if any of the following registry keys have been modified after the InstallShield prerequisite has been run:

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Session Manager\FileRenameOperations

The registry keys are counted before and after the InstallShield prerequisite is run. If these numbers are not the same, it is assumed that the file is trying to restart the system and exit the installation.

Return Code Specified in the .prq File

If the exit code of the .exe file matches the return code specified in the InstallShield prerequisite file (.prq), the target machine may need to be restarted. The return code of the .prq file is entered in the **Specify the return code the application returns if a reboot is required** box of the InstallShield Prerequisite Editor. For more information, see Specifying Parameters for Installing an InstallShield Prerequisite.

Return Codes 1641 and 3010

The target machine may need to be restarted if a reboot state is returned for an .msi package that is run for an InstallShield prerequisite. The standard Windows Installer reboot return codes are:

- ERROR_SUCCESS_REBOOT_INITIATED (1641)
- ERROR_SUCCESS_REBOOT_REQUIRED (3010)

Behavior Specified in the .prq File

One of the fields on the Behavior tab in the InstallShield Prerequisite Editor enables you to specify what should happen if any of the aforementioned conditions are or are not applicable to a particular InstallShield prerequisite. For more information, see Specifying the Behavior for an InstallShield Prerequisite that Requires a Restart.

Specifying Installation Behavior for an InstallShield Prerequisite

When you are creating a new InstallShield prerequisite or modifying an existing one, you can specify whether end users may skip the InstallShield prerequisite installation. You can also specify how the prerequisite installation should proceed if it appears that the target machine does or does not need to be restarted. The Behavior tab of the InstallShield Prerequisite Editor is where you specify these types of InstallShield prerequisite installation behavior.

Specifying that an InstallShield Prerequisite Requires Administrative Privileges

In the Windows Vista and later environment, end users typically run as standard users without administrative privileges. If an InstallShield prerequisite requires administrative privileges, Windows displays a User Account Control (UAC) prompt that requires end users to provide consent or credentials.

If your InstallShield prerequisite does require administrative privileges or if you want the InstallShield prerequisite to be installed per machine, you can specify that on the Behavior tab of the InstallShield Prerequisite Editor.

Task ***To specify that an InstallShield prerequisite installation requires administrative privileges:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Behavior** tab.
3. Select the **The prerequisite requires administrative privileges** check box.

Note • *This setting applies to installations that are run on Windows Vista and later systems. Earlier versions of Windows ignore this setting.*

Allowing End Users to Choose Whether to Install an InstallShield Prerequisite

You may want your InstallShield prerequisite's installation to display a message box that permits end users to choose whether they want to install the InstallShield prerequisite. You can indicate this on the Behavior tab of the InstallShield Prerequisite Editor.

Task ***To allow end users to choose whether to install an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Behavior** tab.
3. Select the **The prerequisite may be optionally skipped by the user** check box.

Specifying Whether to Include the Name of a Prerequisite in the List of Setup Prerequisites to Be Installed on the Target System

If a target system needs one or more setup prerequisites to be installed, the setup prerequisite dialog is typically displayed at run time before the main installation runs. When an end user clicks the Install button on this dialog, the setup prerequisites are installed.

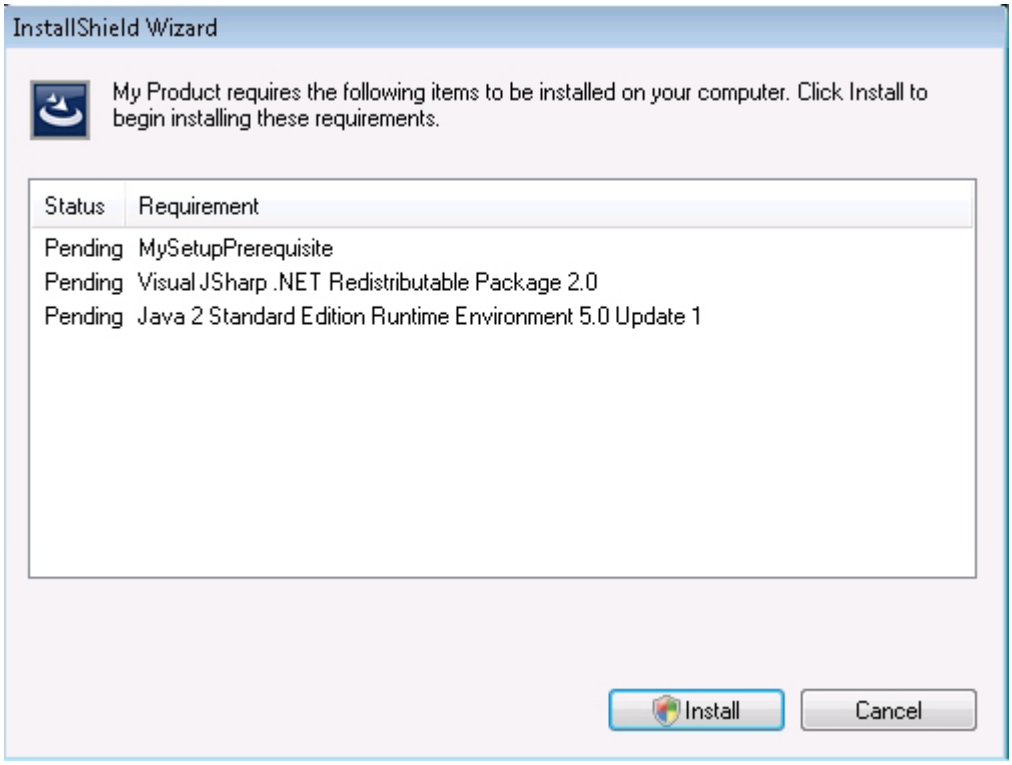


Figure 6-1: Sample Setup Prerequisite Dialog that Shows the List of Setup Prerequisites that Need to Be Installed

The Behavior tab of the InstallShield Prerequisite Editor lets you specify whether you want a setup prerequisite to be hidden from the list in the prerequisite dialog. If a prerequisite is hidden, it is installed when the conditions require it, even though it is not listed as one of the prerequisites that needs to be installed.

If all of the setup prerequisites in an installation are hidden, the installation does not display the setup prerequisite dialog at run time.

Task ***To specify whether to list a setup prerequisite in the setup prerequisite dialog at run time:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Behavior** tab.
3. Do one of the following:
   - To hide the setup prerequisite, select the **The prerequisite should be hidden from the installation list** check box.

     Note that if the setup prerequisite is hidden, it is launched with its silent command-line parameters—not its standard command-line parameters. To learn more, see Specifying Command-Line Parameters for an InstallShield Prerequisite.
   - To include the setup prerequisite in the list, clear the **The prerequisite should be hidden from the installation list** check box.

***Note** • Note that feature prerequisites are never listed in the setup prerequisite dialog at run time, regardless of whether the **The prerequisite should be hidden from the installation list** check box is selected or cleared. That is, if you add an InstallShield prerequisite to your project and associate it with a feature, that feature prerequisite is not listed in the setup prerequisite dialog that is displayed at run time before the main installation runs. For more information about feature prerequisites, see Setup Prerequisites vs. Feature Prerequisites.*

Specifying Whether to Show the Progress of an InstallShield Prerequisite Installation at Run Time

The Behavior tab of the InstallShield Prerequisite Editor lets you specify whether you want the InstallShield prerequisite installation to show the status bar that shows the actual progress of the prerequisite installation, along with installation progress messages from Windows Installer, at run time. This functionality is available only if the prerequisite launches an .msi file; it is not possible if the prerequisite launches a **Setup.exe** file. In addition, the installation that contains the InstallShield prerequisite must be a Basic MSI or InstallScript MSI installation.

If the progress is not shown—or if the installation that contains the InstallShield prerequisite is an InstallScript installation, the prerequisite installation displays an **Installing [PrerequisiteName]** message on the run-time dialog, and the status bar loops continuously until the prerequisite installation has completed.

Task ***To specify whether to show the progress messages and the status bar for an InstallShield prerequisite installation:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Behavior** tab.
3. Do one of the following:
   - To show the progress, select the **Progress should be shown in the prerequisites window (raw MSI files only)** check box.
   - To avoid showing the progress, clear the **Progress should be shown in the prerequisites window (raw MSI files only)** check box.

***Note** • If you specify that the progress should be shown, only some command-line parameters are supported. Command-line parameters are specified on the Application to Run tab of the InstallShield Prerequisite Editor. For more information, see Specifying Command-Line Parameters for an InstallShield Prerequisite.*

Also note that if you specify that the progress should be shown for a prerequisite that is an .msi package, the user interface of the prerequisite's .msi package is not displayed by default. If you want to override this behavior, you can specify `/qf` as a command-line parameter on the Application to Run tab of the InstallShield Prerequisite Editor.

Planning for Issues that Could Occur with an InstallShield Prerequisite Installation

Sometimes after an InstallShield prerequisite installation has been run, the conditions still indicate that the InstallShield prerequisite needs to be installed. If that occurs, either of the following may be true:

- The InstallShield prerequisite was not successfully installed.
- The conditions that were specified for the InstallShield prerequisite were not accurate.

For example, a condition may indicate that the InstallShield prerequisite needs to be installed if a particular file does not exist on the target machine. If the file is still missing even after the InstallShield prerequisite is installed, it is possible that the condition should not have been created. The Behavior tab of the InstallShield Prerequisite Editor lets you indicate how the installation should proceed under these types of circumstances.

Task ***To plan for potential issues related to an InstallShield prerequisite's installation:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Behavior** tab.
3. In the **If, after installing the prerequisite, the conditions still indicate it is required** list, click the appropriate item.

Specifying the Behavior for an InstallShield Prerequisite that Requires a Restart

When you are creating a new InstallShield prerequisite or modifying an existing one, you can specify how the InstallShield prerequisite installation should proceed if it appears that the target machine does or does not need to be restarted. For example, in some cases, you may want the installation to first prompt the end user before restarting the machine; in other cases, you may want to skip the restart. The Behavior tab of the InstallShield Prerequisite Editor provides the flexibility needed to carry out the response that is appropriate for each InstallShield prerequisite.

Task ***To specify the behavior for an InstallShield prerequisite that requires the target machine to be restarted:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Behavior** tab.
3. In the **If the prerequisite requires a reboot** list, click the appropriate item.

To learn about the available options, see Behavior Tab.

Specifying Dependencies for an InstallShield Prerequisite

When you are creating a new InstallShield prerequisite or modifying an existing one, you can specify other InstallShield prerequisites (.prq files) on which this InstallShield prerequisite depends. The Dependencies tab of the InstallShield Prerequisite Editor is where you specify the dependencies.

Adding a Dependency for an InstallShield Prerequisite



Task ***To add a dependency for an InstallShield prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Dependencies** tab.
3. Click the **Add** button. The **New Dependency** dialog box opens.
4. In the **File** box, type the path and name of the .prq file that is required for the current InstallShield prerequisite. To find the .prq file by browsing, click the ellipsis (**...**) button. InstallShield prerequisite files are stored in the following location:

   *InstallShield Program Files Folder***SetupPrerequisites**
5. Click **OK**.

Modifying a Dependency for an InstallShield Prerequisite



Task ***To modify an existing dependency setting:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Dependencies** tab.
3. Select the dependency that you would like to modify.
4. Click **Modify**. The **New Dependency** dialog box opens.
5. Modify the setting as needed.

Removing a Dependency from an InstallShield Prerequisite



Task ***Remove a dependency from a prerequisite:***

1. In the **InstallShield Prerequisite Editor**, open the prerequisite that you want to modify.
2. Click the **Dependencies** tab.
3. Select the dependency that you would like to remove.
4. Click **Remove**.

Saving an InstallShield Prerequisite

Once you have created a new InstallShield prerequisite or modified an existing one using the InstallShield Prerequisite Editor in InstallShield, you must save the InstallShield prerequisite file (.prq).



Task ***To save an InstallShield prerequisite that is open in the InstallShield Prerequisite Editor:***

1. On the **File** menu, click **Save As**. The **Save As** dialog box opens.
1. Specify the file name and .prq extension for the InstallShield prerequisite file. InstallShield prerequisite files are stored in the following location:

   *`InstallShield Program Files Folder`***\SetupPrerequisites**
2. Click the **Save** button.

InstallShield prerequisites that are stored in the SetupPrerequisites directory are listed in the Redistributables view.

Designing Merge Modules

Merge modules enable several development teams to work on separate components independently. Each team can populate its installation database without immediately considering how their work will affect other components. When the components are fully developed, each component in the merge module database can be merged into the main product installation database.

InstallShield enables you to create your own merge modules that can be used in any of your installation projects or distributed for use by other installation developers.

The most important step in creating your merge module lies in its design. To have a well-designed merge module, you need to break up your files into components, which constitute the developer's perspective of the project. Unlike creating a complete installation with features, components, and files, a merge module contains only components, which, in turn, contain files.

The major aspects involved in creating a merge module are listed below.

Table 6-1 • Major Aspects of Merge Module Creation

Merge Module Element	Description
Components	Enable you to group your application data together. They constitute the developer's view of a project—containing data such as files, registry entries, and shortcuts. The Components view is where you design the components for your merge module.
Files, registry data, and shortcuts	Elements that are added to a component to complete the hierarchy, as described above.
Advanced settings	Enable you to publish your component, register COM servers, file extension servers, and MIME types. You can also use the component's advanced settings to create an application paths entry in the registry.

Creating a Merge Module Project

A merge module (or .msm file) contains all of the logic and files needed to install distinct pieces of functionality. For example, many applications require Visual Basic run-time DLLs. Instead of having to include the file in a component and figure out its installation requirements, you can simply attach the Visual Basic Virtual Machine merge module to one of your project's features.

Merge module properties include the following:

- Product Name
- Product Version
- Language
- INSTALLDIR

You can specify any merge module dependencies and exclusions in the Module Dependencies and Module Exclusions settings in the General Information view.

To create a merge module, start by creating a Merge Module project.

Task ***To create a Merge Module project:***

1. On the **File** menu, click **New**.
2. Click the **Windows Installer** tab.
3. In the box of project types, click **Merge Module Project**.
4. In the **Project Name** box, enter a name for your project.

   Important • *When you create a new Merge Module project, the name that you specify in the New Project dialog box should have 35 characters or fewer. This is because the name that you enter is used with the GUID in the ModuleID field of the* ***ModuleSignature*** *table of your merge module file, and the limit for the object name portion of the ModuleID field is a maximum of 35 characters.*
5. In the **Location** box, specify the location where you want to save the project, or click the **Browse** button to navigate to the location.
6. Click **OK**.

Specifying the Default Destination Folder for a Merge Module

The value that you enter for the INSTALLDIR setting in the General Information view serves as the default folder for all of your merge module's files. That is, its value is assigned to the Windows Installer folder property **INSTALLDIR**, which is the default component destination folder.

If you would like to let the users of your merge module override the default destination directory, enter **[TARGETDIR]** in the INSTALLDIR setting. To learn more, see Overriding a Merge Module's Destination.

Other Destination Folder Considerations

Note that each component in your Merge Module project also has a Destination setting. The component's Destination setting overrides the INSTALLDIR setting in the General Information view. Therefore, if you want all of your merge module's files to be installed by default to the merge module's destination folder, enter [INSTALLDIR] for all of your components' Destination settings.

Selecting the Merge Module Language

The Language setting in the General Information view lets you to select the language for which your module is designed. For example, if your module installs a user interface, specify the language for which that user interface was designed. If the module is run on a machine that does not match your module language setting, it is not installed.

Tip • *Select Language Independent to allow your module to run on all languages.*

Adding Dependencies to Merge Modules

While some merge modules are self-contained, some require other merge modules in order to function properly. For example, your merge module may require the Visual Basic run-time libraries. If this is the case, you need to add that merge module as a dependency to your new merge module.

When adding a file to a merge module, you may encounter a Setup Best Practices violation if the file is contained in another merge module that is stored on your system. If this happens, you should add that merge module as a dependency.



Caution • *When you are creating a typical Basic MSI or InstallScript MSI project, merge modules are automatically added to your project when you add a file that is available within a merge module. When you are creating a merge module and you add a file that is available within another merge module, you will receive a Setup Best Practices alert telling you to set that merge module as a dependency. InstallShield does not create the dependency for your Merge Module project.*

Dependency information entered in the General Information view is written to your project's **ModuleDependency** table, which you can view using the Direct Editor.

Adding Dependencies from the Redistributables Gallery

A merge module is in your redistributables gallery if it is located in one of the paths that are specified on the Merge Modules tab of the Options dialog box.



Task ***To add one or more dependencies from the redistributables gallery:***

1. In the View List under **Installation Information**, click **General Information**.
2. In **Module Dependencies** setting, click the ellipsis button (**...**). The **Module Dependencies** dialog box opens.
3. Select the check boxes of the merge modules that are required for the merge module that you are creating.
4. Click **OK**.

InstallShield adds the dependency to the grid in the General Information view. You can change the settings for the dependency if necessary.

Adding Dependencies that Are Not in the Redistributables Gallery



Task ***To add a dependency that is not present on your machine:***

1. In the View List under **Installation Information**, click **General Information**.
2. In **Module Dependencies** setting, click the **Add a new module dependency** button.

InstallShield adds the dependency to the grid in the General Information view. Configure each of its settings as needed.

Note • *Be careful when entering the value for the Module ID setting, since InstallShield does not check to ensure that it is correct. You will not know if it fails until you try to run the installed program.*

Adding Exclusions to Merge Modules

Occasionally you may need to exclude certain merge modules from being associated with the module that you are creating. This may occur because an older version of a module might not be compatible with your new module. As a result, you need to exclude that module from any setup that your new module will be associated with.

Adding Exclusions from the Redistributables Gallery

A merge module is in your redistributables gallery if it is located in one of the paths that are specified on the Merge Modules tab of the Options dialog box.

Task ***To add an exclusion from the redistributables gallery:***

1. In the View List under **Installation Information**, click **General Information**.
2. In **Module Exclusions** setting, click the ellipsis button (**...**). The **Module Exclusions** dialog box opens.
3. Select the check boxes of the merge modules that are not compatible with the merge module that you are creating.
4. Click **OK**.

InstallShield adds the exclusion to the grid in the General Information view. You can change the settings for the exclusion if necessary.

Adding New Exclusions that Are Not in the Redistributables Gallery

Task ***To add an exclusion that is not present on your machine:***

1. In the View List under **Installation Information**, click **General Information**.
2. In **Module Exclusions** setting, click the **Add a new module exclusion** button.

InstallShield adds the exclusion to the grid in the General Information view. Configure each of its settings as needed.

Note • *Be careful when entering the value for the Module ID setting, since InstallShield does not check to ensure that it is correct. You will not know if it fails until you try to run the installed program.*

Publishing a Merge Module to a Repository from Within a Merge Module Project

If you are designing a merge module that you would like to reuse in Windows Installer–based installations or share with other users, you can publish it to a repository. InstallShield enables you to configure a release for the merge module so that the merge module is published to a repository whenever the release is built.



Task ***To configure the settings for publishing a merge module to a repository from within the merge module project:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the release that you would like to configure.
3. Click the **Events** tab.
4. In the **Publish Merge Module** setting, select the **Publish to repository** option.
5. In the **Repository Name** setting, select the appropriate repository. All of the repositories to which you have access are listed in this setting.
6. In the **Display Name** setting, specify a name for the merge module that you are publishing to the repository. This name is displayed for the merge module in the Redistributables view.
7. In the **Publisher** setting, type your name.
8. In the **Description** setting, type a description of the merge module.

Whenever you build the selected release, InstallShield publishes the merge module to the specified repository.

Merge modules are built into an installation at build time. If you make a change to a repository merge module and then republish it to the repository, any project that has the old version of the repository merge module will be updated the next time that a release of the project's installation is rebuilt.

Creating InstallScript Objects

Project *• This information applies to InstallScript Object projects.*

InstallShield enables you to create your own InstallShield objects that can be used in any of your installation projects or distributed for use by other installation developers. Objects replace the templates found in earlier versions of InstallShield.

Note *• Objects that you create can be used only with InstallShield X or later, InstallShield DevStudio, or InstallShield Professional 6.1 or later.*

Creating an object is little different from creating a standard installation project. One difference is the importance of scope. When you create an object, you need to ensure that the object's scope is optimal for its intended use. The scope of an object affects its usability. If the scope is too narrow, the object may be insufficient for your needs. If the scope is too large, the object may to be too cumbersome to be practical.

The following sections describe the steps in the object creation process.

Creating an Object

Currently, there are two ways to create an object. You can create a new InstallScript Object project from the New Project dialog box or convert an existing InstallScript project into an InstallScript object.

Task ***To convert an existing InstallScript project into an InstallScript object:***

1. On the **File** menu, click **Save Project As**. A **Save As** dialog box opens.
2. Select a folder in which to create a copy of the project. Note that you cannot save the new object project in the same folder as the original project.
3. In the **File name** box, type a name for the new project.
4. In the **Save as type** list, select **InstallScript Object Project (*.ism)**.
5. Click **Save**.

An object project file (.ism file) with the name that you specified, and all project subfolders, are created in the location that you specified. That project opens in InstallShield.

Object Design

The purpose of an object is to install a discrete piece of functionality. For example, you may need to install the latest version of DirectX in order for your application to run. Rather than including all the necessary files separately in your installation project, you can include the DirectX object that comes with InstallShield. Imagine if this same object contained, for example, Visual Basic run-time .dll files and MDAC support. The object would be too large for your needs. It would install files that are not needed and would increase the time required for the installation process. Therefore, it is a good idea to streamline your object as much as possible. Do not try to include four different technologies in one object. Instead, create four separate objects.

Objects follow the same structure as traditional installations, with one major difference—objects do not support setup types. The highest level of organization in an object is the feature. Under features fall components. Additionally, objects do not support shell objects. If you would like to create shell objects from an object, you will need to do so via script.

For more information, see Designing Objects.

Building an Object

Building an object is very similar to building an installation project. The easiest way to complete this task is to use the Release Wizard. The main difference between building an object and building an installation project is the fact that you do not need to specify a media type for an object. The wizard enables you to create media in the CD-ROM format only.

For more information, see Building an Object from the User Interface.

Testing Objects

As with any other software endeavor, it is a good idea to test before you release to the public. Testing an object is easy—just create a standard installation project, add your object to it, build it, and run it. If it works as planned, you are ready to include your object in projects. If not, you will need to debug. See Testing and Debugging an Object. (Note that if you have the object project and its associated installation project open in separate instances of InstallShield and you modify and rebuild your object, you must close and reopen the associated installation project to incorporate those modifications.)

Distributing Objects

As used here, *distributing an object* does not mean including that object in an installation and installing the files from that object on a target machine. Instead, *distributing an object* refers to creating an installation that will make that object available on other users' machines for use in their installation projects. Because you do not want to install the logic contained within an object, you cannot just add it to a component as you normally would. To help you register your objects, you can use the InstallShield Object Installer object. When you associate this object with a feature, you will be prompted to select the object from your local gallery that you would like to distribute.

For more information, see Distributing an Object.

Designing Objects

The design of your object—how components and file groups are arranged—accompanied by the actual content of the object, is the single biggest hurdle you need to clear in order to have a stable, usable object.

Several key areas are involved in designing an object. Each is described below.

Object Project Settings

Another difference between standard installation projects and objects lies in the project settings. Tabs for object creation are available on the Project Settings dialog box. On these tabs, you can define the language the object was developed in, the display name of the object, and the icon and help topic associated with the object. You can also can choose the kind of wizard interface, if any, that you would like to provide with your object.

For more information, see Designing an Object's Wizard.

Managing Components

It is important that you take the time to properly divide your project into features. When designing your features, you need to plan for scenarios such as localization and conditional selection of features. For more information, see Designing Features for Objects.

Properties and Methods

Properties and Methods allow your users to interact with your object both during design time and run time. For more information, see Creating a Property and Creating a Method.

Creating a Design-Time Wizard

If you include any properties in your object, it is a good idea to provide a graphical interface so that users of your object can easily set those properties according to their needs. For more information, see Designing an Object's Wizard.

Creating the Runtime User Interface

Your object can display dialogs to the end users of installations that include the object. For information on creating a run-time user interface for your object and calling that interface from an installation that includes the object, see Creating the Object's Run-Time User Interface.

Localizing Your Object's Design Environment

The InstallShield interface is localized into English and Japanese. If you would like your object to be localized to these languages, see Localizing an Object's Design-Time Environment for more information.

Internationalizing Your Object's Run-Time Experience

If your application is designed to run in multiple languages, you will need to prepare an internationalized object. Installing language-specific files with an object differs slightly from standard installations. For more information, see Preparing an Object for Internationalization.

Making Your Object Compatible with Installations During Run Time

Your object should support all the operating systems and languages that are supported by any installation that includes the object. For more information, see Making Your Object Compatible with Installations During Run Time.

Designing Features for Objects

It is very important that you design your features well. A haphazard design of features may make it difficult for you to internationalize your object, selectively build files into the object, or conditionally install files. The following sections discuss a few guidelines that will help you accurately and efficiently design the feature layout of your object.

Selectively Installing an Object's Features

More often than not, you will want to place conditions on the selection of your object's features. For example, if you are creating an object that installs DCOM, you will want to check if the target system already has DCOM installed. If DCOM is present, reinstalling it would be a waste of time and may cause complications. To allow your object's features to be conditionally selected without unwanted effects on the project that includes the object, you must design your object according to the following guidelines.

Your object should never select its own features but only deselect them as needed. (An object's features are selected by default.) If your object selects one or more of its own features, that will cause the feature that includes the object to be selected—even if that feature had previously been deselected by the end user or the project's internal logic.

However, an object's deselecting its own features can also potentially cause a problem. The following paragraphs describe the problem and its solution.

Suppose a project has two main features, A and B; B has no subfeatures but contains an object that, in turn, contains a feature, C. The InstallScript engine automatically deselects a feature when all its child features are deselected; so if feature C is not selected, the DCOM object will not be selected. If the DCOM object is not selected, feature B will not be selected. This behavior can cause problems if you have components associated with the main feature, in this case feature B. Because the feature that contains the components is not selected, the components directly associated with it will not be installed.

If you want your component to be selected regardless of whether DCOM is selected, create a "dummy" feature within your object. If feature C is not selected, the dummy feature will still be selected. Since the dummy feature is selected, feature B will also be selected, and its component will be installed.

Installing Language Dependent Files

When developing an installation that targets multiple languages, you often have different language versions of the same file. If these files have a common name, you may encounter problems when InstallShield builds your installation and copies all of the installation files to the release. When the Release Wizard runs, it takes all of the files within a feature and stores them in the same directory on the release. If you have two files with the same name, but in separate components, one of those files will be overwritten if the two components belong to the same feature. To overcome this problem, you will need to create a separate feature for each language that you are targeting. For more information, see Preparing an Object for Internationalization.

Managing Object Files

When you build an object, all of its files are sent to the release in an uncompressed fashion. As a result, objects that contain multiple files with the same name can run into problems. If those files with the same name reside in the same feature, only one of them will end up in the release. The rest of the identically named files will be overwritten by the last one sent to the release.

The main reason you would have files with the same name is for multiple language support. If that is the case, it is recommended that you create a separate feature for each language that you are supporting. For more information, see Preparing an Object for Internationalization.

An additional layer of assurance can be added by defining a CD-ROM folder for every feature. By defining a CD-ROM folder, you are specifying where on the media the files for each feature will be sent. To define a CD-ROM folder, simply navigate to the Features view and enter a name for your folder in the CD-ROM Folder property.

You must define a CD-ROM folder for every feature that contains a file with the same name as any file that the build places uncompressed in the Disk1 image folder (for example, the compiled script file **Setup.inx**, or any file that you placed in the Support Files/Billboards views' Advanced Files node's Disk1 subnode).

Designing an Object's Wizard

When you include properties in your object, it is a good idea to provide an interface for the users of your object to modify those properties. InstallShield can create a stock wizard for you based on the properties that you created, or you can create your own wizard using Visual Basic or Visual C++.

The InstallShield Stock Wizard

The InstallShield stock wizard is well-suited for in-house use of your object. The stock wizard displays all of your read-only properties and provides an edit field for your read/write properties.

There are a few limitations of the stock wizard that can be overcome by using a custom wizard. For example, the stock wizard cannot be customized to fit your needs. You cannot pick and choose which properties you would like to display, nor can you customize the way the wizard looks. If you would like to create a customized interface for your wizard, you will need to create your own wizard in Visual C++ or Visual Basic. Another drawback of the stock wizard is that it does not provide support for array properties. Although the stock wizard has a few limitations, it is simple to implement and use.

Custom Wizards

If you need to provide a more professional look to your wizard, greater functionality, or customized display, you should create your own wizard using Visual C++ or Visual Basic. If you had these applications installed before you ran the InstallShield installation, the functionality to create an InstallShield object wizard will have been added. If you installed either of these programs after you installed InstallShield, you will need to run the InstallShield installation in maintenance mode in order to add the wizard functionality to your Visual Studio application. To create an InstallShield object wizard in Visual C++, click New on the File menu. Then, select InstallShield Object Wizard from the list of project types. In Visual Basic, select InstallShield Object Wizard from the New Project dialog box.

When you create an InstallShield Object Wizard project in Visual C++, a new ATL-MFC based VC wizard framework project will be displayed. In Visual Basic, a template project will be created. Commented code within the each project type shows how to retrieve and save the object properties. For more information, see Custom Object Wizards.

When you are finished creating your wizard, compile it.



Note *• When you are distributing an object that has a custom wizard, make sure to include any dependencies for the wizard .dll file in the distribution installation project. Include the required files in a feature that you place above the InstallShield Object Installer feature in the Setup Design or Features view. You can avoid having dependencies by creating a static wizard .dll file.*

Specifying Whether to Use the InstallShield Stock Wizard or a Custom Wizard



Task ***To specify whether you want to use the InstallShield stock wizard or a custom wizard that you have created:***

1. In the View List under **Installation Information**, click **General Information**.
2. Do one of the following:
   - If you want to use the stock wizard: In the **Object Wizard** setting, select **Use InstallShield Object Stock Wizard**.
   - If you want to use your own custom wizard: In the **Object Wizard** setting, click the ellipsis button (**...**). The **Browse for My Custom Object Wizard** dialog box opens. Select the wizard file that you created for your object.

When you build a release for your project, the object will include the wizard that you specified.

Custom Object Wizards

To use your own wizard for your object, rather than InstallShield's predefined stock wizard, you must supply an in-process COM server. This server must implement an IDevWizard interface. This interface is called by InstallShield when the object is first included in a project, and when Modify is selected from the context menu of an already included object in the Objects view's Features pane. This interface is useful for displaying values of object properties and allowing those values to be changed.

This interface must offer a Display method with two arguments:

- An IDispatch pointer that provides access to the object's properties.
- A Boolean return value that indicates whether changes should be made to an existing object's properties or, if there is no existing object, whether a new object should be created.

In C++, this interface is defined as follows:

```
interface IDevWizard : IDispatch
{
HRESULT Display([in] IDispatch *pObject,
                [out, retval] VARIANT_BOOL* Confirmed);
};
```

The server can also implement an ISetupDesignObjectBuild interface. The InstallShield release builder calls this interface to determine which of the object's features should be excluded from the build of a project containing the object. (If this interface is not implemented, all of the object's features are included in the build.)

This interface is useful for excluding from the built project those object features that will never be installed because of selections that the installation author made in the object's wizard. For example, if your object offers optional support for various drivers, and the project author chooses not to include some of those drivers, the features containing those drivers can be excluded from the built project.

This interface, if implemented, must offer an IsIncluded method with three arguments:

- An IDispatch pointer that provides access to the object's properties.
- A string value specifying the feature, for example, **Feature 1\Subfeature 1A**.
- A Boolean return value that indicates whether the component should be included in the build.

In C++, this interface is defined as follows:

```
interface ISetupDesignObjectBuild : IDispatch
{
HRESULT IsIncluded([in] IDispatch *pObject,
                   [in] BSTR FeaturePath,
                   [out, retval] VARIANT_BOOL* Included);
};
```



Note *• When you are distributing an object that has a custom wizard, make sure to include any dependencies for the wizard .dll file in the distribution installation project. Include the required files in a feature that you place above the InstallShield Object Installer feature in the Setup Design or Features view. You can avoid having dependencies by creating a static wizard .dll file.*

Localizing an Object's Design-Time Environment

For each of the languages in which the InstallShield interface is localized, you can specify a different display name, short name, .htm file, and icon for your object. You perform this localization on the Project Settings dialog box's Language-Specific Object Properties tab.

The first option on the Language-Specific Object Properties tab is the IDE Language list box. Select the first language you would like to support and, if it is not English, uncheck the Use Default box. Then, fill in the rest of the Language-Specific Object Properties Tab.

If you are only supporting one language, you can close this dialog box. However, if you plan on supporting additional languages, you can select the next language from the list and provide the necessary information for that language.



Note *• If you create your own wizard, you will need to provide run-time strings in the wizard's executable file for each language that you are supporting.*

Preparing an Object for Internationalization

Designing an object that installs different files based on the target machine's language is similar to designing an installation with the same functionality. One vital difference between objects and standard installation projects makes internationalizing your object slightly more complex than installations. This difference—the fact that the release builder places an object project's files uncompressed in the disk image—requires you to handle your internationalized files at the feature level. The following sections discuss the steps and strategies required to prepare your object for internationalization.

Adding Languages to Your Object

If you have the Premier edition of InstallShield, you can associate any of the supported languages with your object through the Language tab of the Project Settings dialog box.

Designing Language-Specific File Groups

The most important step in creating an internationalized object is separating your language-specific data into components. This process is best illustrated with an example. If you are supporting three languages, English, French, and German, and you would like English to be the default language, you should have a minimum of four components. The first component will be for English-only files. The Language property for this component should be set to English and all the languages that you are not directly supporting. By doing this, you are enabling your object to be installed on any language, not just the three you support. If the target machine is running in a language other than one of your three supported languages, the English files will be installed.

For the French and German components, simply add the French and German files to their respective components. Set the language of the French component to French and set the language of the German component to German.

Finally, add your language-independent files to another component. Mark this component as language independent. The table below should help to illustrate the layout of your four file groups.

Table 6-1 • Sample File Group Layout

Language of files	Component language
All English-specific files	English + all unsupported languages
All French-specific files	French
All German-specific files	German
All language-independent files	Language independent

Designing Language-Specific Features

If you would like to include language-specific data in your object, you need to create a feature for each of the languages that you are targeting. This step may seem superfluous, since you cannot actually define a feature's language. However, due to the way files are stored in the disk image, you cannot add components from multiple languages to one feature. When the Release Wizard runs, it takes all the files within a feature and stores them in the same directory on the release. If you have two files with the same name, but in separate components, one of those files will be overwritten if the two components belong to the same feature. To overcome this problem, you will have to make a separate feature for each language that you are targeting. Then, simply add the corresponding components.

Making Your Object Compatible with Installations During Run Time

Your object should support all of the operating systems and languages that are supported by any installation that includes the object. Operating system mismatches could cause the installation to not perform correctly; language mismatches can confuse the installation's end users. To help avoid such mismatches, do one of the following:

- Add support to your object for all operating systems and languages that are supported by InstallShield. To add operating system support to your object project, go to the General Information view and configure the Platform Filtering setting. To add language support to your object project, see Preparing an Object for Internationalization.
- Include, with the object, documentation that informs installation authors which operating systems and languages are supported. To do this, create an .htm file that contains the necessary information, then on the

Project menu, click Settings. Click the Language-Specific Object Properties tab, and specify the .htm file in the HTML Help field.

Properties and Methods

By definition, properties are used to describe attributes associated with a data structure, and methods are procedures that are executed when an object receives a message.

Throughout InstallShield, you can add, create, and set properties associated with different aspects of your installation. Properties can be useful in object scripts .

Methods, like properties, allow your object to interact with the installation that contains it. Methods, for all practical purposes, are functions. As a matter of fact, the only difference between a method and a function is the fact that a method is declared with the method keyword. Methods allow you to expose your object's functions to other objects and installations.

Creating a Property

Properties can be useful in object scripts. In an object script, properties enable your object's users to change certain settings of your object during either design time through the use of an InstallShield Object Wizard or during run time by passing parameters through script. Common properties include passing error messages from the object to the installation, getting the user name, or any information that your object needs during either run time or design time. For examples of how properties are used, see the inline help for any of the InstallShield objects included in your Objects view.

Note the following details about properties:

- If a Boolean property's value is set to TRUE, either by the object script or the script of the installation that includes the object, subsequent checks of the property's value will show the value to be –1 rather than 1.
- You cannot pass a LIST variable between an object and an installation script; you can, however, pass a string array.

Creating a Property



Task ***To create a property:***

1. Open the **InstallScript** view.
2. In the **InstallScript** explorer, right-click **Properties** and click **Add New Property**. The **Add New Property** dialog box opens.
3. Define your property's settings.
4. Click **OK**.

The Add New Property dialog box adds the following to your script:

- A property declaration with the following syntax:

```
property(<procedures>) <property data type> <property name> ( );
```

<procedures> is **get** for a read-only property, **put** for a write-only property, and **get,put** for a read/write property. For example:

```
property(get,put) STRING MyProperty ( );
```

You can specify arguments for your property by adding the arguments' data types inside the parentheses after the property name. For example:

```
property(get,put) STRING MyProperty ( NUMBER );
```

If you add argument data types to the property declaration, you must add corresponding argument variables to the definitions of the property's procedure functions.

- A declaration of a variable to store the property's value within the script. For example:

  ```
  STRING m_strMyProperty;
  ```

  If you specify an array property in the Add New Property dialog box (by selecting the Array check box), two variables are declared: a variable of type VARIANT that stores the property's value and an unsized array variable that is used in the **InitProperties** function block to initialize the property's value. For example:

  ```
  STRING arrayMyArrayProperty();
  VARIANT m_arrayMyArrayProperty;
  ```

- A statement, in the **InitProperties** function block, that initializes the variable that stores the property's value. For example:

  ```
  m_strMyProperty = "My Default Value";
  ```

  If you specify an array property in the Add New Property dialog box (by selecting the Array check box), the variable that stores the property's value is assigned the value of an array variable. For example:

  ```
  m_arrayMyArrayProperty = arrayMyArrayProperty;
  ```

  To initialize elements of the array property to non-null or nonzero values, add assignment statements for the array variable's elements before the statement above. For example:

  ```
  /* Resize the array variable, which is declared with no size. */
  Resize ( arrayMyArrayProperty , 3 );

  /* Assign values to the array variable's elements. */
  arrayMyArrayProperty(0) = "zero";
  arrayMyArrayProperty(1) = "one";
  arrayMyArrayProperty(2) = "two";

  m_arrayMyArrayProperty = arrayMyArrayProperty;
  ```

- A function call, in the **ReadProperties** function block, to retrieve the property values stored in the property bag object. This call has the following syntax:

  ```
  <Read xxxx Property> ( PropertyBag, "<property name>", <property value variable> );
  ```

  <Read*xxxx*Property> is the Read*xxxx*Property function that is appropriate to the data type of the property's value. For example:

  ```
  ReadStringProperty( PropertyBag, "MyProperty", m_strMyProperty );
  ```

- A function call, in the **WriteProperties** function block, to save the property values to the property bag object. This call has the following syntax:

```
<Writexxxx Property> ( PropertyBag, "<property name>", <property value variable> );
```

<Write*xxxx*Property> is the Write*xxxx*Property function that is appropriate to the data type of the property's value. For example:

```
WriteStringProperty( PropertyBag, "MyProperty", m_strMyProperty );
```

- A function block or blocks defining the property's procedures—that is, defining the actions that are executed when the property's value is read or written to by a project containing the object or the object's design-time wizard (if any)

  A **get_<property name>** function is defined for a read-only or read/write property; a **put_<property name>** function is defined for a write-only or read/write property. For example:

```
function STRING get_MyProperty()
begin
    return m_strMyProperty;
end;

function void put_MyProperty(newVal)
begin
    m_strMyProperty = newVal;
end;
```

  If you specify arguments for your property (by adding the arguments' data types to the property declaration, inside the parentheses after the property name), you must add corresponding argument variables to the definitions of the property's procedure functions. For example:

```
function STRING get_MyProperty( szAddedArgument )
function void put_MyProperty( szAddedArgument, newVal )
```

  When you are adding arguments to the put_ function, place them before the newVal argument. The newVal argument takes the value that is written to the property by an assignment statement in the calling project or by the object's design-time wizard (if any).



Note • *The get_ and put_ functions must not be renamed; if they are renamed, your script will not function properly.*

Creating a Structure-Valued Property

To create a property whose value is a data structure, add the following to your object script:

- A global declaration of the data structure; for example:

```
typedef STRUCT
begin
    NUMBER Mem1;
    NUMBER Mem2;
end;
```

- A declaration of a read-only property whose data type is OBJECT. (To write values to the property, you will define a method.) For example:

```
property(get) OBJECT StructProp();
```

  Place this declaration in the Properties Section block of the object script.

- A declaration of a variable to store the property's value within the object script. For example:

```
STRUCT m_structStructProp;
```

Place this declaration in the Local Variables Section block of the object script.

- A declaration of a method with an argument of the appropriate data type corresponding to each member of the data structure. For the STRUCT structure defined above, the following is an example of an appropriate method declaration:

  ```
  method NUMBER SetStructProp(NUMBER, NUMBER);
  ```

  Place this declaration in the Methods Section block of the object script.

- Statements, in the **InitProperties** function block, that initialize the members of the variable that stores the property's value. For example:

  ```
  m_structStructProp.Mem1 = 3;
  m_structStructProp.Mem2 = 5;
  ```

- Function calls and statements, in the **ReadProperties** function block, to retrieve the property member values stored in the property bag object. The function calls have the following syntax:

  ```
  <Readxxxx Property> ( PropertyBag, "<storage name for member value>", <member value variable> );
  ```

  <Read*xxxx*Property> is the Read*xxxx*Property function that is appropriate to the data type of the property member's value. For example:

  ```
  ReadNumberProperty(PropertyBag, "StructProp_Mem1", nStructPropMem1);
  ```

  For each such function call, add below it a statement that sets a member of the property variable equal to the value that the function retrieved. For example:

  ```
  m_structStructProp.Mem1 = nStructPropMem1;
  ```

- Function calls, in the **WriteProperties** function block, to save the property member values to the property bag object. The function calls have the following syntax:

  ```
  <Writexxxx Property> ( PropertyBag, "<storage name for member value>", <property variable member> );
  ```

  <Write*xxxx*Property> is the Write*xxxx*Property function that is appropriate to the data type of the property member's value. For example:

  ```
  WriteNumberProperty(PropertyBag, "StructProp_Mem1", m_structStructProp.Mem1);
  ```

- A get_<property name> function block defining the property's get procedure—that is, defining the actions that are executed when the property's value is read by a project containing the object or by the object's design-time wizard (if any). For example:

  ```
  function OBJECT get_StructProp()
  begin
      return m_structStructProp;
  end;
  ```

- A function block defining the method that you declared; this method sets the members of the property variable equal to the arguments passed to the method. For example:

  ```
  function NUMBER SetStructProp( nMem1, nMem2 )
  begin
      m_structStructProp.Mem1 = nMem1;
      m_structStructProp.Mem2 = nMem2;
  end;
  ```

Passing a String Array Between an Object and an Installation

To pass a string array between an object and an installation:

- Add the following to the object script:
  - A declaration of a read-write property whose data type is variant. For example:

    ```
    property(get,put) variant Test();
    ```

    Place this declaration in the Properties Section block of the object script.
  - Declarations of the necessary variables, including a variable to store the property's value within the object script:

    ```
    STRING arrayTest();
    variant m_arrayTest;
    ```

    Place these declarations in the Local Variables Section block of the object script.
  - Statements, in the InitProperties function block, that initialize the members of the variable that stores the property's value. For example:

    ```
    m_arrayTest = arrayTest;
    ```
  - A function call, in the ReadProperties function block, to retrieve the value stored in the property bag object. For example:

    ```
    ReadArrayProperty(PropertyBag, "Test", m_arrayTest);
    ```
  - A function call, in the WriteProperties function block, to save the value to the property bag object. For example:

    ```
    WriteArrayProperty(PropertyBag, "Test", m_arrayTest);
    ```
  - A function block defining the property's get procedure—that is, defining the actions that are executed when the property's value is read by a project containing the object or by the object's design-time wizard (if any). For example:

    ```
    function variant get_Test()
    begin
       return m_arrayTest;
    end;
    ```
  - A function block defining the property's put procedure—that is, defining the actions that are executed when the property's value is written to by a project containing the object or by the object's design-time wizard (if any). For example:

    ```
    function void put_Test(newVal)
    begin
       m_arrayTest = newVal;
    end;
    ```
- Add the following to the installation script:
  - In each function block in which you will read a string array from or write it to the object, declarations of an object variable and a string array. For example:

    ```
    OBJECT oObject;
    string szTest(3);
    ```

- A call to GetObject to get a reference to the object. For example:

```
set oObject = GetObject("Object");
if (!IsObject(oObject)) then
    MessageBox( "Failed to get object reference.", INFORMATION );
    abort;
endif;
```

- To set the value of the object's string array property, use code like the following:

```
oObject.Test = szTest;
```

  To retrieve the value of the object's string array property, use code like the following:

```
szTest = oObject.Test;
```

Accessing an Object's Properties

There are two different points in time when your properties can be changed, after you have compiled your object. The first is when your object is added to an installation project. If you provide a wizard interface for your object, its read/write properties can be changed during an installation's design time. The second way to change a property is during run time through script. In order to allow the users of your object to access its properties during run time, you need to document those properties in the object's help topic. With that documentation, your users should be able to change those properties as needed.

Accessing a Property from the InstallShield Interface

When you access an object's properties from within your installation project, you are actually running portions of that object's script right in the InstallShield interface. For example, when you associate an object with your installation project, the object's properties are read with the **get_PropertyName** function that is built into every object containing properties. Alternatively, when you change an object's properties through the object's wizard, the **put**_*PropertyName* function is called, where *PropertyName* is the name of the property that you are changing.

Accessing a Property Through Script

To access an object's properties, you must first get a reference to that object. Then, you need to specify which of the object's properties you would like to access. The following code illustrates how to return an object's status property:

```
function OnBegin()
   /* Declare an object variable to hold a reference to the object. */
   OBJECT oObject;
begin
   /* Get a reference to a custom object named "My Custom Object".*/
     set oObject = GetObject("My Custom Object");

   /* Check the Status.Number property. */
   if oObject.Status.Number != OBJ_STATUS_SUCCESS then
      /* Respond to the error as appropriate for your setup.
      For example, you can display the Status.Description
      text and abort the setup. */
      MessageBox ( oObject.Status.Description, SEVERE );
      abort;
   endif;
end;
```

When you change a property at run time, that information is not stored for future uses of the installation. For example, if the installation is later run in maintenance mode, that property will not retain the value it was given during the initial installation.

If a Boolean property's value is set to TRUE, either by the object script or the script of the installation that includes the object, subsequent checks of the property's value will show the value to be –1 rather than 1.



Tip • *If you simply want to retrieve the current status of an object (that is, the current value of Status.Number) in the InstallScript Object, you can use the* ***GetStatus*** *function. For more information, see GetStatus.*

Creating a Method

Methods, like properties, allow your object to interact with the installation containing it. Methods, for all practical purposes, are functions. As a matter of fact, the only difference between a method and a function is the fact that a method is declared with the method keyword. Methods allow you to expose your object's functions to other objects and setups.

Creating a Method



Task ***To create a method:***

1. Open the **InstallScript** view.
2. In the **InstallScript** explorer, right-click **Methods** and click **Add New Methods**. The **Add New Methods** dialog box opens.
3. Define your method's settings.
4. Click **OK**.

The Add New Method dialog box adds the following to your object script:

- A method declaration with the following syntax:

  ```
  method <method data type> <method name> (<argument data types>);
  ```

  For example:

  ```
  method STRING MyMethod( NUMBER, BOOL );
  ```

- A function block defining the method. For example:

  ```
  function STRING MyMethod( /*NUMBER*/ arg1, /*BOOL*/ arg2 )
  begin
  end;
  ```

Using Methods

An object's methods can be accessed only at run time. You can call an object's methods from within your installation project or from another object. To gain access to an object's methods, you first need to get a reference to that object. The following code displays the current status description for the object:

```
Set oObj = GetObject("ATL 3.0")
MessageBox(oObj.Status.Description, INFORMATION);
```

Object Status Properties

Every InstallScript object—whether predefined or user-defined—has the following read-only properties:

- **Status**
- **StatusBase**
- **StatusStruct**

StatusBase and **StatusStruct** are intended only for advanced users. See the following sections for information on these properties.

Status

The **Status** property is a structure with the following properties as members:

Table 6-2 • Members of the Status Property

Property	Description
Status.Number	The numeric object status.
Status.Description	A string describing the object's status.
Status.szSource	For an error, the source of the error; typically the name of the object as read from one of the object's string entries.
Status.szScriptFile	For an error, the name of the script file that was executing when the error occurred.
Status.nScriptLine	For an error, the line of script that was executing when the error occurred.
Status.nScriptError	For an error, the error return code.

In an object script, you can set these values by calling **SetStatusEx**. When you query the status of an object, by default it first queries the status of its subobjects (if any); if one or more of an object's subobjects report an error (that is, have a **Status.Number** property that is less than OBJ_STATUS_SUCCESS), the value of the first failing object's status property is returned as the value of the parent object's status. A user-defined object's script can override this default behavior by explicitly defining a get_Status function (with no arguments).

You can check the value of an object's **Status.Number** property to determine whether the object can be successfully installed—and, if not, what the cause is—and display the value of **Status.Description** to your end user. You can perform this check at any point in a setup; it is recommended that you at least check for success or failure in the **OnBegin** event handler, as in the following example:

```
function OnBegin()
    /* Declare an object variable to hold a reference to the object. */
    OBJECT oObject;
begin
```

```
    /* Get a reference to a custom object named "My Custom Object". */
    set oObject = GetObject("My Custom Object");

    /* Check the Status.Number property. */
    if oObject.Status.Number != OBJ_STATUS_SUCCESS then
        /* Respond to the error as appropriate for your setup.
        For example, you can display the Status.Description
        text and abort the setup. */
        MessageBox ( oObject.Status.Description, SEVERE );
        abort;
    endif;
end;
```

The following table lists those possible values of **Status.Number** and **Status.Description** that are common to all objects. Some objects also have possible values of these properties that are unique to that object. For lists of those possible values, see the individual object's help page (by selecting the object in the **Objects** view, or in the **Setup Design** or **Features** view under its associated feature).

Table 6-3 • Possible Values of Status.Number and Status.Description that Are Common to All Objects

oObject.Status.Number	oObject.Status.Description
OBJ_STATUS_SUCCESS	The operation was successful.
OBJ_STATUS_OSINVALID	Incorrect target operating system.
OBJ_STATUS_DISKSPACE	Not enough disk space.
OBJ_STATUS_LANGUAGESUPPORT	Language not Supported.
OBJ_STATUS_NOTADMINISTRATOR	Administrator rights required.
OBJ_STATUS_SETKEY	Unable to set registry key.
OBJ_STATUS_GETKEY	Unable to get registry key.
OBJ_STATUS_SETVALUE	Unable to set registry value.
OBJ_STATUS_GETVALUE	Unable to get registry value.
OBJ_STATUS_SETTARGET	Unable to set script-defined folder value.
OBJ_STATUS_LOADDLL	Unable to load DLL.
OBJ_STATUS_FILELOCKED	A required file was locked.
OBJ_STATUS_FILENOTFOUND	File not found.
OBJ_STATUS_FILECOPY	File copy failed.
OBJ_STATUS_FILELAUNCH	Failed to launch file.
OBJ_STATUS_SETLOGDATA	Unable to set data in setup log.

Table 6-3 • Possible Values of Status.Number and Status.Description that Are Common to All Objects (cont.)

oObject.Status.Number	oObject.Status.Description
OBJ_STATUS_GETLOGDATA	Unable to get data from setup log.
OBJ_STATUS_INITVARS	Unable to initialize variables.

StatusBase

The **StatusBase** property is a structure with the same members as the **Status** property. This property is typically used only by objects that have a customized get_Status method; except for special cases a setup should not query this property directly, as it could bypass any custom status handling that the object provides.

The **StatusBase** property enables you to use an object's default status handling, including checking the statuses of the object's subobjects. If you have created a custom get_Status function for your object, you can use the **StatusBase** property within the get_Status function to use the default object status handling, as in the following example:

```
function object get_Status()
    object oThis;
begin
    // Do some custom handling here.

    // Get a reference to this object.
    set oThis = GetObject( "" );

    // Fail if this is not an object.
    if( !IsObject( oThis ) ) then
        return NULL;
    endif;

    // Return the standard status information.
    return oThis.StatusBase;
end;
```

StatusStruct

The **StatusStruct** property is a structure with the same members as the **Status** property. This property is typically used only by objects that have a customized get_Status method; except for special cases, an installation should not query this property directly because it could bypass any custom status handling that the object provides.

The **StatusStruct** property allows you to bypass the default object status handling. You can use this property if you want your object to ignore subobject status values, as in the following example:

```
function object get_Status()
    object oThis;
begin
    // Get a reference to this object.
    set oThis = GetObject( "" );

    // Fail if this is not an object.
    if( !IsObject( oThis ) ) then
        return NULL;
    endif;

    // Return the status information directly.
```

```
    return oThis.StatusProp;
end;
```

Creating the Object's Run-Time User Interface

Your object can display a dialog sequence to end users of an installation that includes the object. To define this dialog sequence, use the object script's UI event handlers—OnFirstUIBefore, OnFirstUIAfter, OnMaintUIBefore, and OnMaintUIAfter. (By default, these handlers do not display any dialogs in an object project.)

In a project containing objects with dialog sequences, you have two alternatives for displaying the objects' dialogs:

- Show all object dialog sequences at one point in the project by calling the **ShowObjWizardPages** function.
- Show each object's dialog sequence whenever you choose by calling any or all of the object's Show*xxxxx*UI*yyyyy* methods (that is, ShowFirstUIBefore, ShowFirstUIAfter, ShowMaintUIBefore, and ShowMaintUIAfter).

The ShowObjWizardPages Function

The **ShowObjWizardPages** function is called by default in each UI event handler of an InstallScript installation project created with InstallShield X, InstallShield DevStudio, or InstallShield Professional 6.1 or later; you can study the use of this function in the default event handlers as a guideline. Calling **ShowObjWizardPages** in a project's UI event handler displays each included object's corresponding UI code. For example, calling **ShowObjWizardPages** in a project's OnFirstUIBefore event handler displays each included object's OnFirstUIBefore code. The included objects' UI event handlers are executed in the order in which the objects appear in the project's Components view.

The end user should be able to advance through the installation's entire dialog sequence by using the Next and Back buttons. To allow this, do what is done in the default UI event handler code: assign each dialog function's return value (NEXT or BACK) to the variable nResult, and call **ShowObjWizardPages** as follows:

```
nResult = ShowObjWizardPages(nResult);
```

The ShowxxxxxUIyyyyy Methods

All user-created objects internally support the methods ShowFirstUIBefore, ShowFirstUIAfter, ShowMaintUIBefore, and ShowMaintUIAfter. When you call an object's Show*xxxxx*UI*yyyyy* method, the object's corresponding UI code is executed. For example, the following call executes the object's OnFirstUIBefore code:

```
nResult = oObject.ShowFirstUIBefore(nResult);
```

As with the **ShowObjWizardPages** function, enable end-user navigation by assigning each dialog function's return value to nResult, and calling Show*xxxxx*UI*yyyyy* methods as in the preceding example.

Object UI Event Handler Code

Create object UI event handler code by modifying the default code. This default code, although it does not display any dialogs, contains the code structure that is necessary to enable end-user navigation.

- To display a single dialog, place the dialog function call immediately after either the FIRST_DIALOG or the LAST_DIALOG label.
- To display two dialogs, place the first dialog function call immediately after the FIRST_DIALOG label and the last dialog function call immediately after the LAST_DIALOG label.

- To display more than two dialogs, place the first dialog function call immediately after the FIRST_DIALOG label and the last dialog function call immediately after the LAST_DIALOG label. For each intermediate dialog, place the dialog function call in a code block structured like the FIRST_DIALOG and LAST_DIALOG blocks.

In each of the above cases, assign the return value from the dialog function call to the variable *nDirection*.

Unsupported Features in Objects

Due to both the purpose and implementation of objects, certain features are not supported when you are creating your object in the InstallShield interface. The following list outlines these unsupported features. For more information on the different design philosophies between objects and installation projects, see Designing Objects.

Table 6-4 • Features that Are Not Supported in Objects

Unsupported Feature	Description
Setup Types	Setup types are not supported in objects. If you are creating an object out of an existing installation project, your setup types will be disregarded.
Media Types	When creating an object, you do not have any choices as to the type of media that you would like to build. All objects will be built uncompressed with the CD-ROM option.
Shortcuts and Folders	The Shortcuts view is not supported in objects. In order to create a shortcut or folder within an object, you need to do so from the script with the **AddFolderIcon**, **CreateProgramFolder**, or **CreateDir** function.
Support Files	Where standard installations allow you to specify splash screens; billboards; and language-dependent, language-independent, and advanced files in the Support Files view, objects allow only language-dependent and language-independent files.
Product Registry Keys	 ***Windows Logo*** • *When any installation created with InstallShield is run, product registry keys are automatically created on the target machine. The product registry keys, which are created at HKLM\SOFTWARE\<Company name>\<Product name>\<Version> are required for Microsoft Logo compliance. Therefore, if you would like to have these registry keys created, you need to add them yourself.*

Building an Object

Building an object is essentially the same as building a standard installation. To build your object from the InstallShield user interface, you will need to use the Release Wizard, which prompts you for all the information necessary to build your object.

Building an Object from the User Interface

Note • *An object must be compiled using the library files* **Ifxobject.obl** *and* **Isrt.obl**. *These files are automatically referenced by your object project if you created your project by using the New Project dialog box's InstallScript Object project type or the Save As command on the File menu. If you created your object project in some other way, click Settings on the Build menu. Then check the Libraries (.obl) box on the Compile-Link tab to make sure that* **Ifxobject.obl** *and* **Isrt.obl** *are listed there.*

Building an object is essentially the same as building a standard installation. To build your object, you will need to use the Release Wizard, which prompts you for all the information needed to build your object. The Release Wizard displays the following panels for object projects:

Table 6-5 • Object-Specific Panels in the Release Wizard

Release Wizard Panel	Description
Specify a Release Panel	Enables you to create a new release or select an existing release.
General Options Panel	Lets you specify preprocessor variable definitions and advanced release properties.
Modify Property Dialog Box/Release Wizard—Platforms Panel	Lets you select the operating systems that your object supports.
Setup Languages Panel	Displayed if more than one language is selected in the Project Settings property sheet's Language page. Lets you select the languages that your object will support, a default object language, and whether to let end users select the language in which the object runs.
Digital Signature Panel	Lets you digitally sign your object. Digitally signing your object assures end users that the code within your object has not been modified or corrupted since publication.
Postbuild Options Panel	Lets you copy disk image folders to a folder or FTP site, or execute a batch file, after the next time you build the release.
Release Settings Summary Panel	Displays a summary of release information. ***Note*** • *When you build an object, that object is automatically added to the Objects view and can be used in any installation that you create from that point on. If a previous version of that object already exists in the gallery, it will be overwritten with the newly built object. If you build more than one media, the latest media will be copied to your Objects view.*

Compiling and Building an Object from the Command Line

Compiling and building an object from the command line is handled in an identical fashion to compiling and building an installation from the command line. To learn how to compile and build your object from the command line, see Using ISCmdBld.exe to Build a Release from the Command Line.

Testing and Debugging an Object

Testing your object is similar to testing a standard installation. The main difference lies in the fact that you must include your object in a standard installation in order to test it. This means you must first build your object, include it in an installation project, and then build and run the installation project. The following methods of debugging are available to you when developing your object.

Note *• If you have the object project and its associated installation project open in separate instances of InstallShield and you modify and rebuild your object, you must close and reopen the associated installation project to incorporate those modifications.*

Script Debugger

The Script Debugger enables you to step through the code of your installation as it runs. To debug an object with the Script Debugger, you must include that object in a standard installation project, compile, and debug. Alternatively, you could design your object as a standard installation, test and debug it, and then convert it to an object through the following procedure.

Task ***To convert an object:***

1. On the **File** menu, click **Save As**. The **Save As** dialog box opens.
2. Select a folder in which to create a copy of the project. Note that you cannot save the new object project in the same folder as the original project.
3. In the **File name** box, type a name for the new project.
4. In the **Save as type** list, select **InstallScript Object Project (*.ism)**.
5. Click **Save**.

An object project file (.ism file) with the name that you specified, and all project subfolders, are created in the location that you specified. That project opens in InstallShield.

For additional information, see InstallScript Debugger.

Note *• When you are designing your object as a standard installation with the intent to later convert it to an object, be aware of the following restrictions:*

- *An object cannot contain setup types; any setup types that you define or setup type functions that you call in the project are ignored when it is converted to an object project.*
- *There are a number of functions and events that are not supported in objects.*
- *All information on built media will be lost when the project is converted; you will need to rebuild the project.*

Debugging an Object on Another Machine

Generate inline debugging information is an option on the Build Settings dialog box's Compile/Link tab. It enables you to debug your object on a machine other than the machine on which the object was created. If you do not use this option, debugging information is placed in a file named **Setup.dbg**, and its location is stored in the header of **Setup.inx**. This scheme works fine if you debug the script on the same machine on which you did the compilation. However, on a different machine, the .dbg file may not be stored in the same location. The **Generate inline debugging information** option was designed to alleviate this problem. If you use this option, all debugging information is stored in **Setup.inx**. This way, the installation never needs to search for the debugging information.

It is recommended, for two reasons, that you clear the **Generate inline debugging information** option before compiling the **Setup.inx** file that you ship with your final product. First, the debugging information increases the size of **Setup.inx** and may cause the installation to run more slowly. Second, inline debugging information will make it easier for others to reverse engineer your code.

Using the InstallShield Cabinet and Log File Viewer

Use the InstallShield Cabinet and Log File Viewer to do the following:

- View an InstallScript cabinet file (.cab) or InstallScript header file (.hdr), as well as their compressed files, registry entries, components, features, and other data. This tool also lets you extract files from the .cab file.
- View InstallScript log files (.ilg) that are created by InstallScript installations. This tool enables you to see what your installation has recorded in the log file, which contains important uninstallation information in a binary format.

To learn more, see InstallShield Cabinet and Log File Viewer.

Debugging a Custom Design-Time Wizard

If you have created a custom design-time wizard for your object, you may need to debug that wizard.



Task ***To debug the wizard, do the following:***

1. From within your development environment, launch a debug version of the wizard, setting breakpoints as desired.
2. In the InstallShield interface, insert the object in a feature of another project to launch the object's wizard.
3. If your development environment is Microsoft Visual Basic, you must also do the following:
   a. Minimize the interface. A **Server Busy** message box opens. (You may need to click the title bar's minimize button a few times before the message box opens.)
   b. In the **Server Busy** message box, click the **Switch To** button or press ENTER.

Distributing an Object

The major advantage of objects is the fact that anyone creating an InstallShield installation can include your object in their installation with a minimal amount of work, as long as your object is installed and registered on their machine. The easiest way to register an object is to use the InstallShield Object Installer. This object enables you to package the object that you would like to distribute, and it will register that object on the target machine, if InstallShield X or later, InstallShield DevStudio, or InstallShield Professional 6.1 or later is present.



Task ***To add the InstallShield Object Installer to your installation:***

1. Open the **Objects** view. The **InstallShield Objects/Merge Modules** pane lists all of the available objects.
2. In the **Features** pane, select the feature to which you want to add the InstallShield Object Installer.
3. Right-click **InstallShield Object Installer** and click **Add to selected feature**.

An associated wizard appears to guide you through the customization process.



Note • *The object that you would like to distribute must be present in your InstallShield Object Gallery. If it is not present, you need to register that object. To learn how, see Registering Objects in InstallScript Projects.*

Object Scripts

InstallShield enables you to extend object support by defining additional functions and constants that are not supported in object projects by adding scripts in the InstallScript view. Essentially, you can write scripts that enhance the functionality supported by the InstallShield interface.

Functions Not Supported in Object Projects

The following is a list of InstallScript functions that are not supported in object projects.

- OnFileReadOnly
- OnRemovingSharedFile
- OnFileLocked
- OnNextDisk
- OnMD5Error
- OnFileError
- OnComponentError
- SdSetupType
- SdSetupTypeEx
- SetupType
- ComponentSetupTypeSet
- ComponentSetupTypeEnum
- ComponentSetupTypeGetData

Constants Not Supported in Object Projects

The following is a list of InstallScript constants that are not supported in object projects.

- IFX_ONNEXTDISK
- IFX_ONNEXTDISK_MSG
- IFX_MAINTUI_MSG
- IFX_ONMAINTUI_CAPTION
- IFX_SDFINISH_MSG1
- IFX_SDFINISH_MAINT_MSG1
- IFX_SDFINISH_MAINT_TITLE

Functions Supported Only in Object Projects

The following is a list of InstallScript functions supported only in object projects.

- SetStatus
- WizardDirection

Adding a ScriptDefinedVar Property to Your Object

Several of the objects included in InstallShield have a ScriptDefinedVar property. This property enables you to set at run time the value of a script-defined variable that you used to define a path (for example, **<MYSERVICEFOLDER>\MyService.exe**) at design time. For example, main script code like the following lets the end user specify the value of *<MYSERVICEFOLDER>*:

```
/* Get a reference to the object that is named
"New Custom Object 1" in the Components pane. */
set oObj = GetObject("New Custom Object 1");

/* Get folder selection from end user. */
svDir = TARGETDIR;
AskDestPath ( "" , "Specify folder for service files:" ,
    svDir , 0 );

/* Tell the object the desired value of <MYSERVICEFOLDER>. */
oObj.ScriptDefinedVar("<MYSERVICEFOLDER>") = svDir;
```



Task ***To include the ScriptDefinedVar property in your object script:***

1. Open the **InstallScript** view.
2. In the **InstallScript** explorer, right-click **Properties** and click **Add New Property**. The **Add New Property** dialog box opens.
3. Make the following entries:
    c. In the **Property Name** box, type the following:

    ScriptDefinedVar

    d. In the **Data Type** list, select **String** (since you will be writing string data to the property).

e. In the **Access Method** list, select **Read/Write**. (You can select **Write Only** instead if you are certain users of the object will never want to check the current value of the property from their main script.)

4. Click **OK** to close the dialog box and automatically add all the necessary code to your object script.

5. Make the following changes to the property script code that was automatically added:

   a. Open the **InstallScript** view.

   b. In the **InstallScript** explorer, under **Properties**, double-click **ScriptDefinedVar** to move the script editor pane's insertion point to the ScriptDefinedVar declaration. Add a string parameter to the declaration—that is, change the declaration from this:

   ```
   property(get,put) STRING ScriptDefinedVar();
   ```

   to this:

   ```
   property(get,put) STRING ScriptDefinedVar( STRING );
   ```

   c. In the **InstallScript** explorer, under **Functions**, double-click get_ScriptDefinedVar to move the script editor pane's insertion point to the **get_ScriptDefinedVar** function block. Find the following function block:

   ```
   function STRING get_ScriptDefinedVar()
   begin
       return m_strScriptDefinedVar;
   end;
   ```

   Change it to this:

   ```
   function STRING get_ScriptDefinedVar( szScriptVar ) /* Add string argument. */
   begin
       return TextSub.Value( szScriptVar ); /* Get value associated with szScriptVar. */
   end;
   ```

   d. In the **InstallScript** explorer, under **Functions**, double-click **put_ScriptDefinedVar** to move the script editor pane's insertion point to the **put_ScriptDefinedVar** function block. Find the following function block:

   ```
   function void put_ScriptDefinedVar(newVal)
   begin
       m_strScriptDefinedVar = newVal;
   end;
   ```

   Change it to this:

   ```
   function void put_ScriptDefinedVar( szScriptVar, newVal ) /* Add string argument. */
   begin
       TextSub.Value( szScriptVar ) = newVal; /* Associate newVal with szScriptVar. */
   end;
   ```

Using System Variables in Objects

System variables are predefined variables that contain information such as the source path, the target path, the Windows folder, and the Windows system folder. You cannot declare these variables in your script. InstallShield automatically initializes system variables when the installation process begins.

Although you can change and initialize system variables within objects, it is not recommended. When you change a system variable in an object, that change will affect the entire installation. For example, if your object initializes **TARGETDIR** to a specific value, that value will override what was previously set in the installation project. Therefore, the entire installation will be installed to the directory specified in the object, and not the directory specified by the installation developer.

Setting Your Object's Destination Through the Script

You may decide to define of one of your object's components' Destination properties in terms of a script-defined variable. If you do, users of your object will need to ensure that the value of the script-defined variable is set before they call **ComponentCompareSizeRequired** in their script; otherwise, the function will report insufficient disk space. You can provide the ability to set the value of the script-defined variable in one or both of the following ways:

- A write-only or read/write object property whose put_ function calls **ComponentSetTarget**, for example:

```
function VOID put_SetTargetDest ( szScriptVar , szValue )
begin
   ComponentSetTarget ( MEDIA , szScriptVar , szValue );
end;
```

- An end-user dialog displayed by the object's OnFirstUIBefore event handler.

7

Modularizing Installation Projects to Distribute Development Work and Enable Reuse

The developer installation manifest (DIM) support in InstallShield is targeted toward engineering teams who want to foster collaboration, enable distributed installation development, and maximize efficiency in their organizations. A DIM is a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete, logically separated portion of a product installation. Some benefits of using DIMs are as follows:

- Working with DIMs enables multiple team members to contribute to the development of the installation simultaneously. Each software developer or other team member can work on a separate DIM that the release engineer can reference in one or more installation projects.
- Release engineers can reuse DIMs in multiple installation projects, enabling efficiency.
- DIMs include support for virtually the same functionality that is available in Basic MSI projects. This gives authors of DIMs all of the flexibility that they need to develop their portions of an installation.

This section of the documentation explains how to work with a DIM project in InstallShield.

Specifying Installation Information for a DIM



Project • *This information applies to DIM projects.*

InstallShield stores your project settings in a single project file (.dim). This file contains all of the information about your DIM project. In the General Information view, you can edit basic information about your DIM—including the author's name, internal notes about the project, and any instructions that you want to include for release engineers to consider when they are including the DIM in installation projects.

Saving a DIM Project File in XML or Binary Format



Project • *This information applies to DIM projects.*

InstallShield lets you save your DIM project file (.dim) in XML or binary format.

Task ***To specify the format of your project file:***

1. In the View List under **Installation Information**, click **General Information**.
2. In **Project File Format** setting, select the appropriate option. Available options are:
   - **Binary**—To save the project file as a database file, select this option. This format is best for the speed of opening and saving the project file.
   - **XML**—To save the project file as a hierarchical text-based format, select this option. This project file format is best for use with source code control systems.

Note • *Your DIM project file (.dim) retains the .ism file extension when you save it in XML or binary format.*

Specifying the Default Destination Folder for a DIM

Project • *This information applies to DIM projects.*

The value that you enter for the INSTALLDIR setting in the General Information view serves as the default folder for all of your DIM's files. That is, its value is assigned to the Windows Installer folder property **INSTALLDIR**, which is the default destination folder. In an installation project, this typically evaluates as follows:

[ProgramFilesFolder]My Company Name`My Product Name`

If you would like to let the users of your DIM override the default destination, enter **[TARGETDIR]** in the INSTALLDIR setting. To learn more, see Overriding the Destination for a DIM Reference.

Using Path Variables in a DIM

Project • *This information applies to DIM projects.*

When you add product files to a DIM project, InstallShield creates a path variable to use as the source of the product files by default. A path variable is a variable that InstallShield uses to represent a directory that contains a project's source files. Note that InstallShield uses path variables at design time and build time; path variables are not available or used at run time when the installation is running on end users' machines.

A general advantage of using path variables instead of hard-coded paths is that if the source files are moved to a different directory on the development system, you need to change only the value of the path variable. That is, you do not need to update each hard-coded path to all of your source files.

Using path variables in DIM projects can help to simplify source file management, especially if the DIM project is located on a different machine than the Basic MSI projects that will include the DIM project.

Path variables can also be helpful if you use source code control to store all of your code and InstallShield project files, and if you have build machines that obtain the appropriate files from source code control, compile the code, and build the installations. In this type of scenario, different release engineers and developers may use different local working directories for obtaining the latest files from source control. Using path variables to define the locations helps to minimize the issues that occur with different source locations.

Predefined Path Variables in DIM Projects

Following is a list of predefined path variables, as well as their typical values, in a DIM project.

Table 7-1 • Predefined Path Variables in a DIM Project

Predefined Path Variable	Typical Value
CommonFilesFolder	**C:\Program Files\Common Files**
ISProductFolder	**C:\Program Files\InstallShield\2016**
ISProjectDataFolder.DIM_GUID	**<ISProjectFolder>\ProjectName**
ISProjectFolder.DIM_GUID	**C:\InstallShield 2016 Projects**
ISRedistPlatformDependentExpressFolder	**C:\Program Files\InstallShield\2016\Redist\Language Independent\i386 Express**
ISRedistPlatformDependentFolder	**C:\Program Files\InstallShield\2016\Redist\Language Independent\i386**
ProgramFilesFolder	**C:\Program Files**
SystemFolder	**C:\Windows\System32**
WindowsFolder	**C:\Windows**

In both cases where the path variable name contains *DIM_GUID*, *DIM_GUID* is the identifier that is defined in the DIM GUID setting of the General Information view of the DIM project. Following are sample path variables in a DIM project:

- **ISProjectDataFolder.8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC**
- **ISProjectFolder.8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC**
- **MyNewPathVariable.8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC**

The GUIDs help to identify the applicable path variables as originating in the DIM project. They also help to minimize or eliminate conflicts that would occur if the same path variable is used in a DIM project and also in a Basic MSI project that contains that DIM, and if different values are assigned to those two path variables.

The path variable names that do not include GUIDs—such as *SystemFolder* and *WindowsFolder*—are not specific to the project that is open in InstallShield, and they are available on all machines. The values of these types of path variables may vary depending on the operating system. For example, on a 32-bit machine, the value of **ProgramFilesFolder** may be **C:\Program Files**. On a 64-bit machine, it may be **C:\Program Files (x86)**.

Syntax for Referencing Path Variables

When a path variable is used in InstallShield, the path variable is enclosed in angle brackets—for example:

<ISProjectDataFolder.8356F8B7_8DE5_4E04_A77A_6FA722CBE1CC>

In some cases, you may want to define a path variable so that it is relative to a different path variable. For example, you may create a standard path variable called **MySourceFiles**. You may also want to use a different path variable called **BinFiles** to define a path to a **Bin** folder inside that **MySourceFiles** directory. When you enter the relative path variable in the Defined Value column for **BinFiles** variable, you must surround the base path variable name with angle brackets. Thus, the Defined Value column for the **BinFiles** variable would be:

<MySourceFiles>\Bin

If the value of **MySourceFiles** is **C:\CheckoutRoot**, the value of BinFiles is **C:\CheckoutRoot\Bin**.

Sample Path Variable Scenarios

If you work in an environment where you create and edit InstallShield projects on one or more machines but you build your installations on other machines, you may need to plan how you want to use path variables in your projects. Otherwise, when you build your installations in InstallShield, you may encounter build errors because of missing files, or the wrong files may be included in your builds.

The following examples illustrate different ways of using path variables in InstallShield projects. The method that is described in example 3 is the easiest one to use.

Example 1: Overriding the Values of DIM Path Variables in the Basic MSI Projects that Contain the DIMs

The following table shows examples of how one predefined path variable and several standard path variables are used in a DIM project.

Table 7-2 • Example 1 Path Variables in a DIM Project

Name	Defined Value	Current Value
ISProjectFolder.4E846FD3_5307_4CF5_9FF1_C476E5505666	(This is a predefined path variable for the folder that contains the InstallShield project file. Its value cannot be manually defined.)	**C:\Source\Units\Utility**
CHECKOUT_ROOT.4E846FD3_5307_4CF5_9FF1_C476E5505666	C:\Source	**C:\Source**
PATH_TO_RELEASE_FILES.4E846FD3_5307_4CF5_9FF1_C476E5505666	<CHECKOUT_ROOT.4E846FD3_5307_4CF5_9FF1_C476E5505666>\SpellLib\Bin\Release	**C:\Source\SpellLib\Bin\Release**

Table 7-2 • Example 1 Path Variables in a DIM Project (cont.)

Name	Defined Value	Current Value
PATH_TO_DICTIONARIES.4E846FD3_5307_4CF5_9FF1_C476E5505666	<CHECKOUT_ROOT.4E846FD3_5307_4CF5_9FF1_C476E5505666>\Dictionaries	**C:\Source\Dictionaries**

In this example, the author of the DIM project assigned the path variable **CHECKOUT_ROOT.4E846FD3_5307_4CF5_9FF1_C476E5505666** to his source root folder—the working directory on his machine where he checks out files from source control—**C:\Source**. Path variables for other folders that contain source files are defined relative to the source root path variable.

If the release engineer uses a different working directory location on a build machine, it may be necessary to override the value of this path variable when building a Basic MSI installation for a project that includes the DIM. Therefore, the release engineer can set a new value for the source root path variable by passing the `-l` parameter to **ISCmdBld.exe** when building the installation from the command line. Since the other path variable values are defined by using the source root path variable, the values of those other path variables are updated accordingly.

Note that using this method may be tedious and error prone, especially if you are building an installation for a Basic MSI project that includes several DIM projects, each with their own path variables that need to be overridden.

Example 2: Changing the Values of the DIM Path Variables According to the Build Environment

If the source root paths on the DIM development machine do not match the corresponding paths on the Basic MSI build machine, you can open the DIM directly and update the values of the path variables as needed. To learn how to open the DIM project from within the open Basic MSI project, see Opening a Referenced DIM Project from Within an Installation Project.

This method may not be optimal if you are trying to deliver daily builds to your testing team and you are still making changes to the DIM projects.

Example 3: Using Path Variables that Are Relative to a Predefined Path Variable Such as ISProjectFolder.DIM_GUID

The easiest way to manage path variables in DIM projects is to always store source files in a path that is relative to a predefined path variable such as **ISProjectFolder. DIM_GUID**. The following table illustrates how you can do this.

Table 7-3 • Example 3 Path Variables in a DIM Project

Name	Defined Value	Current Value
ISProjectFolder.4E846FD3_5307_4CF5_9FF1_C476E5505666	(This is a predefined path variable for the folder that contains the InstallShield project file. Its value cannot be manually defined.)	On a development machine, the current value may be: **C:\Source\Units\Utility** On the build machine, the current value may be: **C:\BuildWorkspace\Source\Units\Utility**

Table 7-3 • Example 3 Path Variables in a DIM Project (cont.)

Name	Defined Value	Current Value
CHECKOUT_ROOT.4E846FD3_5307_4CF5_9FF1_C476E5505666	<ISProjectFolder.4E846FD3_5307_4CF5_9FF1_C476E5505666>\..\.. ***Note*** • *Using the predefined project folder path variable for the value of this path variable, and then using the checkout root path variable for the other path variable values, makes the InstallShield project and all of its source files portable from one machine to another.*	On a development machine, the current value may be: **C:\Source** On the build machine, the current value may be: **C:\BuildWorkspace\Source**
PATH_TO_RELEASE_FILES.4E846FD3_5307_4CF5_9FF1_C476E5505666	<CHECKOUT_ROOT.4E846FD3_5307_4CF5_9FF1_C476E5505666>\SpellLib\Bin\Release	On a development machine, the current value may be: **C:\Source\SpellLib\Bin\Release** On the build machine, the current value may be: **C:\BuildWorkspace\Source\SpellLib\Bin\Release**
PATH_TO_DICTIONARIES.4E846FD3_5307_4CF5_9FF1_C476E5505666	<CHECKOUT_ROOT.4E846FD3_5307_4CF5_9FF1_C476E5505666>\Dictionaries	On a development machine, the current value may be: **C:\Source\Dictionaries** On the build machine, the current value may be: **C:\BuildWorkspace\Source\Dictionaries**

Using the predefined project folder path variable for the value of the checkout root path variable, and then using the checkout root path variable for the other path variable values, makes the InstallShield project and all of its source files portable from one machine to another.

Organizing Files for a DIM

Project • *This information applies to DIM projects.*

In a DIM project, files are organized in a hierarchy. You add the files for your portion of the installation to components. When you include a DIM in a Basic MSI project, you are adding the DIM's components to one or more features in that Basic MSI project. When end users run your Basic MSI installation, they can select which features they want to install, if you allow them to selectively include and exclude features.

When you are adding files to components in DIM projects, you should be aware of Setup Best Practices. By default, the Setup Best Practices Wizard monitors your DIM design and alerts you if you have violated Best Practices.

To learn more about organizing your files, see the following areas of the documentation:

- Including Files and Folders
- Using Components
- Setup Best Practices

Configuring the Target System for a DIM



Project • *This information applies to DIM projects.*

Every installation changes the target system in some way. The simplest installations might only copy files. More in-depth installations make registry changes, edit .ini files, create shortcuts, configure ODBC resources, use environment variables, modify XML files, and modify text files. For more information on how to include these sorts of configuration changes in a DIM project, see the following areas of the documentation:

- Creating Shortcuts and Program Folders
- Editing the Registry
- Changing .ini File Data
- Configuring ODBC Resources
- Using Environment Variables
- Modifying XML Files
- Modifying Text Files

Customizing Installation Behavior for a DIM



Project • *This information applies to DIM projects.*

An important aspect of creating an installation is customizing it for your end users' needs. For example, you may find it useful to create custom actions to add support for something not directly supported by Windows Installer. In addition, you may want to search for installed data on the target system; depending on the results of the search, the installation can proceed as appropriate. Windows Installer properties allow you to set project-wide values for use in your project. For more information on how to customize the installation behavior for a DIM project, see the following areas of the documentation:

- Using Custom Actions

- Searching for Installed Data
- Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties

Configuring Servers for a DIM



Project • *This information applies to DIM projects.*

When you are creating a DIM project, you may find it necessary to provide server-side support for some technology that will be installed on a target system. InstallShield provides support for creating and managing Internet Information Services (IIS) Web sites, managing and organizing SQL scripts by server connections and settings, and managing COM+ applications and components. For more information on how to configure servers for a DIM project, see the following areas of the documentation:

- Configuring SQL Support
- Managing COM+ Applications and Components
- Managing Internet Information Services

8

Updating Applications

Installing upgrades for applications is a much more common operation than installing the original release of an application. This makes creating effective, reliable upgrades a very important task.

"Updating Applications" provides solid background information on the different types of upgrades and also clears up common misconceptions about patching. It helps you determine the best upgrade solution for your product and guides you through the steps of creating upgrades, patches, QuickPatch projects, differential releases, and full releases. In addition, this section explains how you can use FlexNet Connect to notify end users that a new version of your product is ready for release.

Upgrades Overview

The cost of maintaining software applications is often more expensive than the cost of original development. This makes creating effective, reliable upgrades an important task. Being able to distribute robust upgrades for an application depends on how the original installation package was structured and distributed. This help topic will provide a basic foundation for creating robust upgrade packages.

Upgrade Methods for Basic MSI and InstallScript MSI Projects

Windows Installer provides different methods for implementing three different upgrade types: small update, minor upgrade, and major upgrade. Upgrades can be packaged as either a full installation or a patch. A patch is simply another mechanism for implementing a type of upgrade. With a patch, however, you deliver to your customers only the bits required to change an installed file into a new file, unlike a full release.

Upgrade Methods for MSI Database and Transform Projects

MSI Database and Transform projects support Windows Installer major upgrades.

Upgrade Methods for InstallScript Projects

When working with InstallScript installations, you can update an application with either a full release package or differential release. Various considerations such as file size and the presence of files from any previous release must be considered in determining the best upgrade method.

Note *• If you used a version of InstallShield Professional earlier than 6.0 to create media for an InstallScript project, the media cannot be updated.*

Major Upgrades

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

In a major upgrade, the product changes are large enough to merit changes to both the product version number and the product code, as well as the package code. An example is updating version 1.2 of a product to 2.0. A major upgrade acts like a first-time installation if an earlier version is not present. If an earlier version is present, a major upgrade typically uninstalls the earlier version and then installs the new version. It is also possible to have a major upgrade first install over the earlier version and then remove any unused files. This method is more efficient, but strict component-authoring rules must be followed.

How Major Upgrades Work

As the Windows Installer help implies, you must change the product code for the latest version of your installation in order to perform a major upgrade.

Note *• You can set the product code for your installation in the General Information view. It does not matter what the new product code is, as long as it is unique.*

By virtue of having a unique product code, your latest installation will have its own independent registration with the Windows Installer service on end users' machines. This means that unless some other step is taken, the Windows Installer installs the latest version of your product independently of any of the previous versions. This may be okay if the two versions of your product can coexist on the same machine, but then that would not be an upgrade. Therefore, for the sake of this discussion, it is assumed that two versions should not coexist on the same machine.

Once you update the product code for your latest installation, you need to use the Upgrades view to specify information for any previous version or versions that you want to upgrade.

For more information about what is involved in creating a major upgrade, see Creating Major Upgrades.

Major Upgrades at Run Time

When an earlier version of the product is not available on the target machine and the end user runs a major upgrade, the installation behaves as a first-time installation.

If an earlier version of the product does exist on the target machine and the end user runs a major upgrade, they encounter almost the exact same installation experience as if they were to install the latest application on a clean target machine. The difference is that before installing its new resources, the installation typically uninstalls the

earlier version from the target machine. This removal is reflected in the progress bar of the Setup Progress dialog, which gives end users the ability to see the progress of the uninstallation. After the removal of the previous installation has completed, the resources from the latest installation are then installed onto the target machine.

This type of upgrade is a complete uninstallation and then reinstallation of all of the resources associated with your application. Therefore, any data for your application that has been configured by the end user may be completely deleted from the end user's machine. If you need to retain some of the end-user data, create a custom action that backs up this data and then replaces it after the installation of new data has completed.

Minor Upgrades



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

A minor upgrade is a change to the product database and files large enough to merit a change to the **ProductVersion** property but not to the **ProductCode** property. In other words, both the package code and the product version number are different than those in the earlier installation package, but the product code of the minor upgrade does not change. An example is updating version 1.1 of a product to version 1.2. Minor upgrades usually do not have significant changes to the installation organization between versions. A minor upgrade packaged as a full installation acts like a first-time installation if an earlier version is not present, or it installs over an existing installation of a product. Essentially for an upgrade of an existing installation, a minor upgrade installs the differences between your version 1.2 and 1.1 application.



Tip • *You can modify the product version for your upgrade in the General Information view.*

Small Updates



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

In essence, a small update is a type of upgrade used to modify a few files for an installed application; it typically delivers small bug fixes. A small update consists of product changes (such as hotfixes) so small that no change to the product version is necessary or desired. A package code change is required for a small update.

A drawback to small updates installed with versions of Windows Installer earlier than 3.0 is that external programs, including installers for later versions of your product, cannot distinguish between the original version and the updated version. In addition, Windows Installer versions earlier than 3.0 cannot enforce the correct order in which small updates should be applied.

Patching



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Patching is a streamlined mechanism for updating earlier versions of a Windows Installer installation package, thereby updating the application. With a patch, you deliver to your customers only the bits required to change an installed file into a new file. One benefit of a patch is that the size of the upgrade package can be significantly smaller than the full-installation package required to deliver the same upgrade. Keeping an upgrade package as small as possible allows you to more easily deliver your upgrades over the Internet.

However, you should note that a patch may not always present the best solution. For example, if you want to change your installation from compressed to uncompressed, or vice versa, you should package your upgrade as a full installation, but not as a patch. To learn more about determining the best packaging option for your upgrade, see Packaging Options for Upgrades.

A patch is delivered in the form of a patch package (.msp) file, which your end user can apply to an installed product. A patch package is capable of updating as many earlier versions of an installation package as required. A patch package contains separate transforms and instructions for updating each previous version that you specify.

An important aspect of patch creation is generating a patch creation properties (.pcp) file, which defines the parameters on which the patch package is to be created. The .pcp file is a database that has a specific schema, but you can open it and edit it directly with InstallShield or Orca.

The Patch Design view provides an easy-to-use interface to simplify the patch creation process. This view groups together all the logical considerations you will need to make in patch creation.

QuickPatch Projects



Project *• You can create a QuickPatch to update an earlier version of your product only if the earlier version was installed using Windows Installer. The installation for the earlier version of your product must have been created with one of the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

A QuickPatch project is a specific type of project recommended for installation authors who want to ship small, single updates to their users. Changes that are more extensive such as adding custom actions and changing .ini data typically require a standard patch.

QuickPatch authoring provides a simple alternative to creating a patch configuration in the Patch Design view, even though it provides less customization. Fundamentally, both patch creation methods produce the same deliverable types: .msp and .exe files.

With a QuickPatch, you can do any of the following:

- Add new files to the original installation or an earlier QuickPatch.

- Delete files in the original installation.
- Delete files that were added with an earlier QuickPatch.
- Perform the same set of above operations on registry entries.
- Remove custom actions that were included with the original installation but that do not apply to the current QuickPatch project.

The creation of a QuickPatch project always begins with the Create New QuickPatch Wizard. After you have completed the wizard, you can configure project settings once the QuickPatch project opens in InstallShield.

Differential Releases



Project • *This information applies to InstallScript projects.*

In an InstallScript project, you can define and build a differential release—that is, a release that contains the files that are absent from—and/or have a different date, time, size, or attribute than the version in—one or more of a specified set of existing releases. A differential release is used to update the versions of your product that were installed by those existing releases. This type of upgrade also includes all of the maintenance/uninstallation feature's files. The new versions of **Data1.hdr**, **Data1.cab**, and **Layout.bin** do not overwrite the existing versions but are placed in a new folder.



Note • *Since a differential release consists of only the changed files between the specified existing media and the new media, installing a differential release on a system that does not have a specified earlier version of the product would most likely not produce a valid application. To update a system that does not have an earlier version of the product, create a full release.*

Files Included in a Differential Release

When you define a differential release, you specify one or more existing releases to which the current project should be compared when new differential release is created. You specify these existing releases in the Update panel of the Release Wizard. A file in the current project is included in the differential release if any of the following are true:

- The file is absent from—or has a different date, time, size, or attribute than the copy in—at least one of the specified comparison releases.
- The file is in a component whose Difference property is set to Include Always.
- The file is in a component that is absent from at least one of the specified comparison releases.
- The file is in a component whose name is different in at least one of the specified comparison releases.
- The file is in a component whose Destination, Languages, or Operating Systems properties are different in at least one of the specified comparison releases.
- The file is in a component that is associated with a feature whose name or path within the feature hierarchy is different in at least one of the specified comparison releases.

Installing a Differential Release

When an installation initializes and the ADDREMOVE system variable is nonzero, the installation automatically attempts to initialize any differential releases that have been installed with the same product code. The installation enumerates all subfolders of the **DISK1TARGET** folder and searches these subfolders for differential releases; a differential release is found and loaded if all of the following conditions are met:

- The subfolder contains a **Data1.hdr** file that is a valid InstallShield X or later, InstallShield DevStudio 9, or InstallShield Professional 6 or later **Data1.hdr** file for a differential release.
- The product code stored in the header file matches the product code of the installation.
- The product version of the differential release is newer than the product version of the installation.

Full Releases



Project • *This information applies to InstallScript projects.*

With an InstallScript project, you can define and build a full release for an upgrade. A full release includes all of the files linked into the installation project. By default, this type of release behaves as a first-time installation if an earlier version of the product is not already present on the target system. It also is capable of installing over an earlier version and replacing that version completely.

When you create a full release to update one or more earlier versions of a product, you must specify whether the release is version specific or non-version specific:

- **Non-version specific—**This is the default option. A non-version-specific full release can be installed on a system that does not have an earlier version of the product. It can also update any earlier version of the product.
- **Version specific—**A version-specific full release updates only the versions that you specify should be updated. If you set your full release to be version specific but you do not also specify version numbers, the update can be applied to any earlier version of your product.

Determining the Best Upgrade Solution

The methods used to create an upgrade for a product depend on how the original installation package was developed. If the original installation was created with a Windows Installer–based project, you will essentially create a major upgrade, a minor upgrade, or a small update to update your product. If the original installation was created with an InstallScript project, you will create a differential or full release to update your product. Both types of projects are discussed below.

Basic MSI and InstallScript MSI Projects

The first step in creating an installation for any type of upgrade is identifying whether you want to address target systems that do not have any earlier version of your product. Then you can determine what type of method you should use to package the upgrade. The following table presents a general overview to help you decide which method you should use. For a more in-depth discussion of the techniques for packaging the upgrade, see Packaging Options for Upgrades.

Table 8-1 • Possible Upgrade Solutions for Windows Installer–Based Projects

Status of the Target Systems	Type of Installation Needed	Technique for Packaging the Update
Some of the target systems have an earlier version of the product, and some do not have any version of the product.	If file size is not an issue, you can create one installation that does both of the following: • Behaves as a first-time installation if an earlier version of the product is not already present on the target system • Updates an existing product if it is already installed on the target system	Create a major or minor upgrade and package it as a full installation.
Some of the target systems have an earlier version of the product, and some do not have any version of the product.	If you want a small installation for end users who need to update an earlier version of the product, you can create two separate installations: • A full installation that behaves as a first-time installation. • A smaller installation that updates one or more earlier versions of an already installed product. Since this installation consists of only the changed data between the versions to be updated, it usually enables you to deploy your upgrade using much less bandwidth than that required to deploy a full-installation package.	For the first-time installation, create a major or minor upgrade and package it as a full installation. For end users who have an earlier version of your product, create a major or minor upgrade and package it as a patch.  ***Tip*** • *In some cases, a patch is not appropriate. For more guidelines on determining whether a patch is appropriate for your upgrade, see Packaging Options for Upgrades.*  ***Note*** • *A patch package is not a type of upgrade; it is simply a mechanism for distributing an upgrade with a small footprint.*

Table 8-1 • Possible Upgrade Solutions for Windows Installer–Based Projects (cont.)

Status of the Target Systems	Type of Installation Needed	Technique for Packaging the Update
All of the target systems have an earlier version of the product.	You can create a small installation that updates one or more earlier versions of an already installed product. Since this installation consists of only the changed data between the versions to be updated, it usually enables you to deploy your upgrade using much less bandwidth than that required to deploy a full-installation package.	Do either of the following: • Create a major or minor upgrade and package it as a standard patch. • Create a QuickPatch project. The QuickPatch technology in InstallShield enables you to create simple patches for previously installed products. Although customization is somewhat limited when you create a patch by using a QuickPatch project, the creation of a QuickPatch project is a simpler alternative to creating a standard patch. For more details, see Patch vs. QuickPatch Project. ***Note*** • *Like a standard patch, a QuickPatch is a mechanism for distributing an upgrade with a small footprint.*

InstallScript Projects

If the original installation was created with an InstallScript project, you can create one of two types of update releases:

- **Differential release—**This type of release contains the files that are absent from—and/or have a different date, time, size, or attribute than the version in—one or more of a specified set of existing releases. A differential release is used to update the versions of your product that were installed by those existing releases.
- **Full (non-differential) release—**This type of release behaves as a first-time installation if an earlier version of the product is not already present on the target system. It also is capable of installing over an earlier version and replacing that version completely.

Since a differential release consists of only the changed files between the versions to be updated, it usually enables you to deploy your update using much less bandwidth than that required to deploy the full release. However, only a full release can be installed on a system on which no version of your product is currently installed. For more details, see Differential Releases and Full Releases.

Major Upgrade vs. Minor Upgrade vs. Small Update

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Windows Installer supports three types of product upgrades: major upgrades, minor upgrades, and small updates. The following table serves as a guide to help you determine which type of upgrade best suits your needs. If any one of the requirements for your upgrade is not appropriate for a minor upgrade or a small update, you should create a major upgrade. If you are not sure which type of upgrade you should use for your Windows Installer–based project or if you do not have a preference, you can create an automatic upgrade.

Table 8-2 • Major Upgrade vs. Minor Upgrade vs. Small Update

Requirement for the Upgrade	Use a Major Upgrade?	Use a Minor Upgrade?	Use a Small Update?	Note
Change the name of the .msi package	Yes	No	No	The default file name is taken from the Product Name property, provided the .msi file is not compressed in a **Setup.exe** installation launcher.
Enable end users to install earlier versions and the latest version on the same machine	Yes	No	No	
Add a new subfeature	Yes	In some cases	In some cases	If the new subfeature consists of new components only, you can use a small update, a minor upgrade, or a major upgrade. If the new subfeature consists of existing components, you must use a major upgrade.
Move or delete a feature in the product tree	Yes	No	No	
Add a new component to a new feature	Yes	Yes	Yes	
Add a new component to an existing feature	Yes	Yes, if the version of Windows Installer is 2.0 or later	Yes, if the version of Windows Installer is 2.0 or later	Windows Installer 1.x requires new components in an upgrade package to be placed in new features for minor upgrades and small updates; it also requires special command-line handling.
Move or delete a component in the product tree	Yes	No	No	
Change the component code of an existing component	Yes	No	No	

Table 8-2 • Major Upgrade vs. Minor Upgrade vs. Small Update (cont.)

Requirement for the Upgrade	Use a Major Upgrade?	Use a Minor Upgrade?	Use a Small Update?	Note
Change the key file of a component	Yes	No	No	
Add, remove, or modify any of the following: files, registry keys, or shortcuts	Yes	Yes	Yes	If the file, registry key, or shortcut is in more than one component and the component is shared by two or more features, a major upgrade must be used.
Start with an MSI Database or Transform project	Yes	No	No	

Codes Associated with the Different Types of Upgrades

Several Windows Installer codes help identify a product:

- **Package Code**—Part of the Summary Information Stream, the package code identifies a particular database. The package code is not a Windows Installer property. Any two .msi databases with identical package codes must have identical contents. Therefore, you should change the package code for each build.
- **ProductVersion**—This is a Windows Installer property that identifies the product version. Note that Windows Installer uses only the first three fields of the **ProductVersion** property for version comparisons. For example, for a product version of 1.2.3.4, the 4 is ignored. (Note that this is true for comparisons of **ProductVersion** values, and not for file versions.)
- **ProductCode**—This is a Windows Installer property that identifies the GUID of a product. Windows Installer treats two products with different **ProductCode** GUIDs as unrelated, even if the values for the **ProductName** property are the same.
- **UpgradeCode**—This is a Windows Installer property that identifies the GUID that represents a related set of products. The **UpgradeCode** should be consistent across different versions and languages of a family of related products so that it can be used to search for related versions of the product that are already installed. You can set the **UpgradeCode** for an upgrade in the Upgrades view.

For any type of upgrade, you must change various combinations of the package code, product version, and product code to identify the product being installed. The following table identifies when each code should be changed for different types of upgrades.

Table 8-3 • Codes that Need to Be Changed for the Different Types of Upgrades

	Package Code	Product Version	Product Code	Upgrade Code
Small Update	X			
Minor Upgrade	X	X		
Major Upgrade	X	X	X	

Automatic Upgrades



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

To simplify the upgrade creation process, InstallShield offers you the ability to create automatic upgrades. An automatic upgrade is a special type of upgrade that you can create if you do not know which type of upgrade you should use or if you do not have a preference. With automatic upgrades, InstallShield determines which type of upgrade—major upgrade or minor upgrade—would be optimal based on comparisons between the earlier and later packages that you specify.

Patch vs. QuickPatch Project



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

To update a product without distributing a complete, updated installation package, you can create a Windows Installer patch package (.msp). InstallShield provides two different patch creation alternatives serving different scenarios: one method uses the Patch Design view, and the other uses the QuickPatch project type. Find the appropriate solution for your product by reading about both alternatives below.



Note • *Major upgrades are typically not packaged as patches. Therefore, the table below assumes that the patch created through the Patch Design view is for a minor upgrade or a small update. QuickPatch projects produce QuickPatch packages that serve as minor upgrades.*

Table 8-4 • Patch vs. QuickPatch Project

Requirement for the Patch	Use a Patch?	Use a QuickPatch?	Note
Ability to apply many cumulative patches that update a base package	Yes	Yes, if you are using QuickPatch streamlining	If you are not using QuickPatch streamlining, you cannot patch more than 15 times. For more details, see Specifying Whether to Streamline the QuickPatch Package.
Add a new subfeature	Yes	No	

Table 8-4 • Patch vs. QuickPatch Project (cont.)

Requirement for the Patch	Use a Patch?	Use a QuickPatch?	Note
Add a new component to a new subfeature	Yes	No	
Add a new component to an existing feature	Yes, if the version of Windows Installer is 2.0 or later	No	Windows Installer 1.x requires new components in an upgrade package to be placed in new features for minor upgrades and small updates; it also requires special command-line handling.
Add, modify, or remove a file	Yes	Yes	A new target destination cannot be used for a new file in a QuickPatch; only destinations that were previously defined in the original installation can be used for new files. This limitation does not exist for patches that are created through the Patch Design view.
Add, modify, or delete registry data	Yes	Yes	All new registry data being added with a QuickPatch must be associated with a feature that already exists in the original installation.
Add, modify, or delete a shortcut	Yes	No	
Add, modify, or delete custom actions	Yes	Can only delete custom actions that were included in the original base installation.	
Add or remove a redistributable	Yes	No	
Add, modify, or remove ODBC resources	Yes	No	
Edit an .ini file	Yes	No	

Table 8-4 • Patch vs. QuickPatch Project (cont.)

Requirement for the Patch	Use a Patch?	Use a QuickPatch?	Note
Edit an .xml file	Yes	No	
Configure server settings such as IIS Web sites, component services, and SQL scripts	Yes	No	
Digitally sign files (such as your application's executable files) in the patch package	Yes	Not automatically	For QuickPatch projects, you must manually sign the individual files and then add them to your project.

Using the Patch Design View

The Patch Design view provides an integrated, visual method for creating patches and selecting the proper settings associated with each patch configuration you create. You can create multiple patch configurations in the Patch Design view. Each patch configuration contains the settings and data required to build a patch.

In most cases, you will begin patch creation in the Patch Design view. If you are looking for a simple patching solution, you can create a QuickPatch project.

Using the QuickPatch Project

The Create New QuickPatch Wizard, which is launched when you create a QuickPatch project, is targeted at installation authors who want to ship small updates to their users. It provides a simpler alternative to patch creation in the Patch Design view.

Although customization is limited when you create a patch by using a QuickPatch project, it produces the same deliverable types as the Patch Design view: .msp and .exe files.

Packaging Options for Upgrades



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you want to prepare an installation that updates a version of your product installed on an end user's machine, you can consider two different methods for packaging your major upgrade, minor upgrade, or small update:

- You can package your upgrade as a full installation that updates an existing product if an earlier version is installed, or behaves as a first-time installation if no earlier version is present.
- You can package your upgrade as a patch that contains only the changed data (.msi data and byte-level file differences) between the versions to be updated.

Full-Installation Packages

Both a minor upgrade and a small update act like first-time installations if an earlier version is not present, or they install over an existing installation of a product. A major upgrade also acts like a first-time installation if an earlier version is not present; however, a major upgrade typically uninstalls any earlier version and then installs the new version.

Patch Packages

Patches enable you to distribute just the bits and portions of the database necessary to update your application's files to a specific version, possibly resulting in a much smaller package than an upgrade packaged as a full installation. This enables you to deploy your upgrades using much less bandwidth than that required to deploy a full-installation package.



***Note** • A patch is not a type of upgrade. Patching is simply a mechanism for distributing a major upgrade, a minor upgrade, or a small update with a small footprint. In fact, creating a patch involves first designing the upgrade and then packaging it as a patch. Before you create a patch, it is recommended that you test the upgrade as a full-installation package.*

Determining the Best Packaging Option for Your Upgrade

The topic Determining the Best Upgrade Solution includes a table that you can review to determine what type of packing option you should use to update an earlier version of your product. In some cases, a standard patch or a QuickPatch may be seem to be the ideal mechanism for packaging your upgrade. However, under certain conditions, you should package your upgrade as a full installation instead of a patch. For example:

- If the target image was created with Windows Installer 1.2 or earlier, and the upgraded image is created with Windows Installer 2.0 or later, you should package your upgrade as a full installation, but not as a patch. Creating patches for packages that span this schema inflection point can be problematic. For more information, see Val0011, which describes the associated validator for upgrading and patching validation.
- If you want your upgrade to move one or more files on the target system from one location to another, you should package your upgrade as a full installation, but not as a patch. If your end users install a patch for an upgrade that moves files on the target system, problems may occur. For example, the patch may not work, a repair to the target system may not work, a subsequent patch may not work, or end users may not be able to uninstall the product. As a workaround, you can create your patch so that it deletes the files in the old location and adds the files to the new location.
- If you want to change your installation from compressed to uncompressed, or vice versa, you should package your upgrade as a full installation, but not as a patch. If you use a patch in this scenario, a repair to the target system may not work, a subsequent patch may not work, or the end user may not be able to uninstall the product.
- If you need to move files from one .cab file to another, or if you need to change the order of files in a .cab file, you should package your upgrade as a full installation, but not as a patch.
- If your original installation had more than 32,767 files but your latest installation has fewer than 32,767 files, a patch will fail. Similarly, if your original installation had fewer than 32,767 files but your latest installation has more than 32,767 files, a patch will fail. In either case, you should package your upgrade as a full installation.

  Note that if both the original installation and the latest installation have more than 32,767 (or both have fewer than 32,767 files), you can package your upgrade as a patch.

- Patches cannot be created for major upgrades of InstallScript MSI projects. Therefore, if you need to distribute a major upgrade for an InstallScript MSI project, you should package it as a full installation.

Differential vs. Full Releases



Project • *This information applies to InstallScript projects.*

If the original installation for your product was created with an InstallScript project, you need to create a differential release or a full release to update your product.

If some of the target systems have an earlier version of the product, and some do not have any version of the product, you can package your upgrade as a full release. This type of release behaves as a first-time installation if an earlier version of the product is not already present on the target system. It also is capable of installing over an earlier version and replacing that version completely.

If all of the target systems have an earlier version of the product, you can package your upgrade as a differential release. This type of release contains the files that are absent from—and/or have a different date, time, size, or attribute than the version in—one or more of a specified set of existing releases. A differential release is used to update the versions of your product that were installed by those existing releases.



Note • *If you used a version of InstallShield Professional earlier than 6.0 to create media for an InstallScript project, the media cannot be updated.*

Working with Upgrades, Patches, and QuickPatch Projects



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Note that MSI Database and Transform projects have support for major upgrades, but not for minor upgrades, small updates, patches, or QuickPack packages.

If you are going to package your upgrade as a full installation or as a standard patch, you begin by opening the latest version of your project and making the necessary changes, such as adding files and registry entries, as needed.

If you would like to package your upgrade as a QuickPatch, you begin by creating a new project—a QuickPatch project. With a QuickPatch project, you identify which earlier releases should be patched by your QuickPatch.

Refer to the topics in this section to learn how to create major upgrades, minor upgrades, small updates, automatic upgrades, patches, and QuickPatch projects. This section also explains how you can validate your upgrade or patch to determine whether your resulting package will perform its actions correctly.

Understanding File Overwrite Rules



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Windows Installer service uses several file overwrite rules, by default, to determine whether a file included in an upgrade should overwrite a file that already exists on the target system. The rules apply when the **REINSTALLMODE** property uses the o setting to install over older files on the target system. To change this behavior, you can replace the o option with one of the following values:

- p—Reinstall only if there is no equivalent file on the target system.
- e—Reinstall if the file is missing or if it is an earlier or equal version.
- d—Reinstall if file is missing or different.
- a—Reinstall all files, regardless of version.

Note that the setting for **REINSTALLMODE** applies to all of the features being installed; it cannot be set for an individual feature. In addition, setting **REINSTALLMODE** to include a will likely cause prompts for the original installation source during the application of a patch.

Upgrade Considerations



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Note that MSI Database and Transform projects have support for major upgrades, but not for minor upgrades, small updates, patches, or QuickPack packages.

Following are some guidelines for creating upgrades.

Updating the Package Code, the Product Version, and the Product Code

For any type of upgrade, you must change various combinations of the package code, product version, and product code to identify the product being installed. For more information, see Major Upgrade vs. Minor Upgrade vs. Small Update.

Upgrade and Patch Optimization

When you are configuring a minor upgrade or a small update, use the patch optimization functionality in your build settings.

Using patch optimization helps you synchronize component names, component GUIDs, **File** table keys, and **Directory** table keys in the upgraded image with those in the target images. This helps to make the smallest possible patches. If you do not use patch optimization, the patch may be the same size as the target .msi package, or it may break Windows Installer component rules.

The patch-creation process uses the **File** table keys to determine if two files are the same file. (The actual file names cannot be used reliably, since a package might contain more than one file with the same name, installed under different conditions.) If you specify the previous package for patch optimization, InstallShield uses identical **File** table keys for identical files.



Tip *• If your package uses dynamic file linking, it is recommended that you use patch optimization when building your upgrade. This keeps the key files consistent across the releases.*

To use the patch optimization functionality, specify the previous Windows Installer package on the Advanced Settings panel in the Release Wizard. You can also specify the previous Windows Installer package through the Previous Package setting on the Build tab for the release in the Releases view.

Adding a Subfeature with a Minor Upgrade

In general, a minor upgrade should not include a new top-level feature. However, new subfeatures of existing features are allowed. If you are adding a new subfeature for a minor upgrade, set two of the subfeature's settings as follows so that they are installed correctly during a minor upgrade:

- **Remote Installation**—Select **Favor Parent**.
- **Required**—Select **Yes**.

The user interface of a minor upgrade does not usually show the feature tree; however, maintenance mode for the updated installation typically does expose the feature tree. If you want the new subfeature to be excluded from the feature tree so that end users cannot deselect it, set the subfeature's Display setting to Not Visible.

Removing Installed Data

If your minor upgrade removes data such as files, registry settings, or .ini files, simply removing the data that existed in the earlier product version from your project will not cause the file to be removed when the upgrade is applied. To learn how to remove such data, see Configuring Minor Upgrades to Remove Installed Data.

Configuring InstallShield to Automatically Determine the Upgrade Type



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

If you do not know which type of upgrade you should use for your Windows Installer–based project or if you do not have a preference, you can create an automatic upgrade. With automatic upgrades, InstallShield determines which type of upgrade—major upgrade or minor upgrade—would be optimal based on comparisons between the earlier and later packages that you specify. The topics in this section explain how to create and work with an automatic upgrade.

Adding an Automatic Upgrade Item



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*



Task ***To add an automatic upgrade item:***

1. Open the **Upgrades** view.
2. In the **Prepare Setup For Upgrade Scenarios** explorer, right-click **Upgrade Windows Installer Setup** and click **Add Automatic Upgrade Item**. InstallShield adds the automatic upgrade item to the explorer.
3. Rename the item so that it reflects the latest version of your installation.

Configuring Automatic Upgrade Item Settings



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you add an automatic upgrade item to your project, you must configure its automatic upgrade item settings. You can configure these settings when you click your new automatic upgrade item in the Prepare Setup For Upgrade Scenarios explorer of the Upgrades view.

In addition, you need to configure general upgrade settings that InstallShield will use to create your upgrade, depending on whether InstallShield determines that a major or a minor upgrade is necessary for your automatic upgrade item. You can configure these general settings on the tabs that are displayed when you click Upgrade Windows Installer Setup in the Prepare Setup For Upgrade Scenarios explorer of the Upgrades view:

- Common tab
- Advanced tab

Converting Automatic Upgrades to Major or Minor Upgrades



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To convert an automatic upgrade item to a major upgrade or a minor upgrade:***

1. Open the **Upgrades** view.
2. Right-click the automatic upgrade item that you would like to convert and then click **Convert to Minor Upgrade Item** or **Convert to Major Upgrade Item**.

Removing Automatic Upgrade Items

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To remove an automatic upgrade item:***

1. Open the **Upgrades** view.
2. Right-click the automatic upgrade item that you would like to delete and then click **Delete**.

Creating Major Upgrades

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Once you have determined that a major upgrade is the best upgrade solution for you, you can begin to create a major upgrade in the Upgrades view. Note that a major upgrade signifies significant change in functionality and warrants a change in the **ProductCode** property; you can update this property in the General Information view.

Note • *When you change the product code, Windows Installer treats your latest and previous product versions as unrelated, even though the* ***ProductName*** *values are likely the same. If you want both versions of your product to be installable on the same system, you can simply change the product code and the main installation directory (often INSTALLDIR).*

Essentially, a major upgrade is two operations rolled into one installation package. The major upgrade either installs the new version of the product and then silently uninstalls the older version or silently uninstalls the older version and then installs the newer version. The sequence of these two separate operations depends on how you configure the installation package of the newer version of the product.

For more information on creating a major upgrade using InstallShield, refer to this section of the InstallShield Help Library.

Adding a Major Upgrade Item



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Task ***To add a major upgrade item:***

1. Open the **Upgrades** view.
2. In the **Prepare Setup For Upgrade Scenarios** explorer, right-click **Upgrade Windows Installer Setup** and click **Add Major Upgrade Item**.

InstallShield adds a new major upgrade item.

Configuring Major Upgrade Settings



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

When you add a major upgrade item in the Upgrades view, a general setting that you can modify is the scheduling of the RemoveExistingProducts action. You can modify this setting in the Upgrades view by selecting Upgrade Windows Installer Setup and clicking the Common tab.

The configurable settings for your major upgrade item are presented in a more user-friendly format on the Common and Advanced tabs than in the **Upgrade** table in the Direct Editor.

On the Common tab for a major upgrade item, you specify the Upgrade Code value of the products that you want to update as well as the versions of products that you want to update. On the Advanced tab, InstallShield by default includes an ISSetAllUsers custom action; this action sets **ALLUSERS** correctly during a major upgrade. (In addition, ISSetAllUsers sets the custom property **IS_MAJOR_UPGRADE** if a major upgrade is taking place.) To specify whether

to include the ISSetAllUsers action, use the General tab of the Options dialog box. You can access this dialog box by selecting Options from the Tools menu. On that tab, you can clear or select the **Automatically create ISSetAllUsers action** check box.

Removing Major Upgrade Items



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*



Task ***To remove a major upgrade item from the Upgrades view:***

1. Open the **Upgrades** view.
2. In the **Prepare Setup For Upgrade Scenarios** explorer, right-click the major upgrade item and click **Delete**.

Creating Minor Upgrades



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Once you have determined that a minor upgrade is the best upgrade solution for you, you can start by adding a minor upgrade item in the Upgrades view. After you build an installation with a minor upgrade item, the build engine performs a series of tests to determine whether that your latest installation can successfully upgrade the previous version of your installation. However, adding a minor upgrade item is not necessarily required for a minor upgrade to function properly. For more details, see Run-Time Behavior for Minor Upgrades.

Any new features that you add to the latest version of your installation must be added as subfeatures of existing features. These new features must have Favor Parent selected for the Remote Installation setting and Yes selected for the Required setting in the Features view.

Adding a Minor Upgrade Item



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To add a minor upgrade item in the Upgrades view:***

1. Open the **Upgrades** view.
2. In the **Prepare Setup For Upgrade Scenarios** explorer, right-click **Upgrade Windows Installer Setup** and click **Add Minor Upgrade Item**.

InstallShield adds a new major upgrade item.

Tip • *If you are creating an upgrade that can update more than one previous version, you can add additional minor upgrade items.*

Configuring Minor Upgrade Settings

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you add a minor upgrade item in the Upgrades view, a general setting you can modify is how a setup launcher (if your build configuration includes a setup launcher) should behave if an earlier product version is present. You can modify this setting in the Upgrades view by selecting Upgrade Windows Installer Setup and clicking the Common tab.

Once you have added a minor upgrade item in the Upgrades view and made sure that it is selected, you can browse for the .msi database of the earlier product version that you want to target in the Setup to Upgrade setting for that minor upgrade item.

You can also specify release flags in the Release Flags setting to exclude the minor upgrade item from the build.

Configuring Minor Upgrades to Remove Installed Data

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

InstallShield includes support for different methods for configuring a minor upgrade to remove installed data. Two methods are discussed in the following sections.

Method 1

To configure a minor upgrade to remove data that was installed during an earlier release, open your InstallShield project in which you are configuring the upgrade, and in the Files and Folders view, delete the file. Then you can create a record in the corresponding Remove tables within the Direct Editor. The Remove tables in the Direct Editor include the **RemoveFile**, **RemoveIniFile**, and **RemoveRegistry** tables.

To remove registry data from a component in a minor upgrade, you should create a record in the **RemoveRegistry** table. Records in the **RemoveRegistry** table describe the registry key and value to remove when the associated component is installed. Unlike the **RemoveFile** table, the **RemoveRegistry** table does not accept an option to remove the specified registry data when the associated component is uninstalled. Instead, you can author a registry value with the **Uninstall entire key** flag. If your component contains a registry value with a hyphen in the Name field and an empty Value field, the specified registry key and all its contents will be removed when the component is removed.

For other types of data, there is usually either an uninstallation flag available in the Windows Installer table or a corresponding uninstallation table. To remove .ini data, for example, there is a **RemoveIniFile** table. For environment variable data, there is a corresponding uninstallation flag.

Note *• If a component is shared with other products, you should change the component code when removing resources. Furthermore, changing an existing component's component code requires a major upgrade. For major upgrades, you do not need to populate the Remove tables since a major upgrade requires a complete uninstallation.*

Method 2

Mark a component as transitive, and set a condition on the component that will evaluate to false when a minor upgrade is applied.

Removing Minor Upgrade Items

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To remove a minor upgrade item from the Upgrades view:***

1. Open the **Upgrades** view.
2. In the **Prepare Setup For Upgrade Scenarios** explorer, right-click the minor upgrade item and click **Delete**.

Run-Time Behavior for Minor Upgrades

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Running a Minor Upgrade with Setup.exe

If you build a release that includes a **Setup.exe** launcher, your latest installation will be minor upgrade enabled. **Setup.exe** can detect when a previous version of your application exists on a target machine. When **Setup.exe** detects a previous version, it will run the rest of your installation in minor upgrade mode.

When end users run a minor upgrade on target systems that have an earlier version of the product installed, the first thing that they see is a message box that announces that an upgrade will occur. This message box specifies the name of the previous product, alerting the end user as to which existing application will be updated. At this point, the end user can stop the installation process if desired. This prompt is followed by a Welcome dialog, and when the end user clicks Next on the Welcome dialog, the setup installs its new resources. While installing its resources, the installation updates the progress bar in the Setup Progress dialog so that end users can track the upgrade process.



***Tip** • In the Upgrades view, you can disable the initial upgrade prompt when you click Upgrade Windows Installer Setup in the explorer and configure the Common tab settings for small updates and minor upgrades.*



***Note** • At run time during a minor upgrade, Windows Installer does not reinstall a component if its key file would be downgraded in the process. Therefore, the key file in the upgrade needs to have a version number that is equal to or greater than that of the key file in the old package. The exception is when the end user sets the **REINSTALLMODE** property to **a**, which forces the installation of all files, regardless of version. For more information about file overwrite rules, see Overwriting Files and Components on the Target System.*

A minor upgrade installs its new resources over the application that currently exists on the target machine. If you want to completely uninstall and then reinstall your application, you should consider a major upgrade.

Running a Minor Upgrade Without Using Setup.exe

If you intend to distribute your installation without including a **Setup.exe** launcher, end users must use the command line to start the installation. For this reason, you should consider using **Setup.exe**; however, you can achieve similar results without it.

The Installer properties **REINSTALL** and **REINSTALLMODE** must be set from the command line to start an installation in upgrade mode. In all but the most advanced scenarios, the property **REINSTALLMODE** should be set to **vomus** and the property **REINSTALL** should be set to **ALL**. A typical command line can look like the following:

```
msiexec.exe /i \product.msi REINSTALLMODE=vomus REINSTALL=ALL
```

If the update contains features that you do not want to update, you should set **REINSTALL** to a comma-separated list of the features that you want to update, as in the following command:

```
msiexec /i \product.msi REINSTALLMODE=vomus REINSTALL=F1,F3,F5
```

The feature names you use in the **REINSTALL** property are case-sensitive.



***Note** • Do not set **REINSTALL** equal to **ALL** for a first-time installation.*

A critical point to note is that you do not want to set these properties on the command line unless a previous version of your installation already exists on the target machine. If you do, the installation will appear to run in minor upgrade mode, and your application files may not be installed. Be sure that your end users will be able to discern when command line should and should not be used.

***Note** • The **REINSTALLMODE** attribute* v *is very important if you are performing a minor upgrade. This parameter tells the installer to refresh its internal setup cache for your product with the new installation package. Without this parameter, your installation will have no knowledge of any of the changes that you have made to the latest installation package.*

Creating Small Updates

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The steps for creating a small update are the same as the steps for creating a minor upgrade with one key exception: the product version should be changed for a minor upgrade, but not for a small update.

If you need to create your upgrades so that they can be applied to your product using versions of Windows Installer earlier than 3.0, it is recommended that you avoid creating small updates. Since small updates do not change the product version, external programs—including installers for later versions of your product—cannot distinguish a product with the small update applied from one without the small update.

Small updates are typically packaged as a patch, not as a full installation. If you are sure that the upgrades that you create will be applied to your product with version 3.0 or later of Windows Installer, you can take advantage of patch sequencing support and use small-update patches to update earlier versions of your product.

Patching Considerations

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Following are some guidelines for creating patches and QuickPatch projects.

Testing the Upgrade Before Creating the Patch

It is recommended that you test your upgrade as a full-installation package before creating the patch. If the full-installation package successfully updates the earlier version of the application, then you can create the patch.

Creating an Update Launcher (Update.exe)

InstallShield lets you specify whether you want your update to include an **Update.exe** update launcher. An **Update.exe** update launcher is required in the following cases:

- You want to automatically update or install the Windows Installer engine on a target system, when necessary.

- You are building a patch for an InstallScript MSI project.
- You want to automatically install the .NET Framework on a target system, when necessary.
- You are building a patch whose product configuration includes support for installing multiple instances of a product and you want the instance selection dialog to be displayed when appropriate.

  To learn more about multiple-instance support, see Installing Multiple Instances of Products and Creating Patches for Multiple Instances of a Product.

The **Update.exe** update launcher is a bootstrap application that manages the aforementioned scenarios.

If you configure InstallShield so that it does not create an **Update.exe** update launcher, it creates an .msp file.

For more information, see:

- Specifying Whether to Build an Update.exe Update Launcher for a Patch
- Specifying Whether to Build an Update.exe Update Launcher for a QuickPatch Package

Optimizing Patches for Large Files

Selecting the **Optimize Patch for Large Files** check box in the Patch Design view results in smaller bit-level patches for application files that are larger than 4 MB.

For QuickPatch projects, each QuickPatch that is built is automatically optimized for large files.

Patches for Compressed Installations

The patch creation process requires an uncompressed release of the previous installations and the latest installation. If the installations are compressed installations, you can use an administrative image of the releases.

InstallShield automatically creates an administrative image for you if you specify an installation that is compressed for a patch configuration in the Patch Design view. InstallShield also automatically creates an administrative image for you if you specify an uncompressed image in a QuickPatch project.

Patches and the REINSTALL Property

If you discover while testing your patch that you are prompted to locate the original installation source, you can sometimes avoid such problems by setting the **REINSTALL** property explicitly to the features that you have modified. This property is set to **ALL** by default.

Self-Registration of COM Servers in a Patch or QuickPatch Package

InstallShield uses InstallShield self-registration (which uses the **ISSelfReg** table) for new COM servers that are marked to be self-registered in a QuickPatch project if all of the following conditions are true:

- The **ISSelfReg** table exists in the upgraded .msi file that is used to create the QuickPatch.
- The ISSelfRegisterFiles custom action is present in the Execute sequence.
- The ISSelfReg option is selected on the Preferences tab of the Options dialog box in InstallShield.

If any of the aforementioned conditions are false, InstallShield uses Windows Installer self-registration (the **SelfReg** table).

To use InstallShield self-registration in a patch if any of the aforementioned conditions are false, use the Patch Design view instead of a QuickPatch project.

Windows Installer 3.0 (and Later) and Patches

Windows Installer 3.0 and later includes many patch-related enhancements. If your patch or QuickPatch will be applied to target machines with Windows Installer 3.0 or later, you can take advantage of those enhancements. For more information, see the following:

- Patch Sequencing
- Patch Uninstallation
- Non-Administrator Patches

Note that Windows Installer 3.0 and later supports major-upgrade patches that do not have any sequencing data. However, the installer does not support major-upgrade patches that have sequencing data, which would be stored in the **MsiPatchSequence** table.

Windows Installer 1.2 and Patches

To create a patch that targets the Windows Installer 1.2 engine, select the appropriate patch configuration in the Patch Design view. Then on the Advanced tab, set the Minimum Windows Installer Version property to **120**.

Support for Patching Multiple Instances of a Product

If you create a patch for a Basic MSI project that includes support for installing multiple instances of a product in the same context on the same machine, InstallShield configures the patch so that end users can apply it to one instance or all of the instances. To learn more, see Creating Patches for Multiple Instances of a Product.

Customizing File Properties for the Update Launcher



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

InstallShield lets you use custom information for the version resources of the **Update.exe** update launcher. The information is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties.



Note • *The Properties dialog box is different on different versions of Windows. For example, on Windows 7 systems, the version resource information is displayed on the Details tab of the Properties dialog box. However, on Windows XP systems, the version resource information is displayed on the Version tab of that dialog box.*

Also note that some versions of Windows do not show some settings on the Properties dialog box.

To see screen shots of the Properties dialog box for a **Setup.exe** *file (which looks the same as the Properties dialog box for an* **Update.exe** *file), see Customizing File Properties for the Setup Launcher.*

Task ***To override the default InstallShield values for the Update.exe settings with your own custom values:***

1. The procedure depends on whether you are creating a standard patch or a QuickPatch package.

   If you are creating a standard patch in a Basic MSI or InstallScript MSI project:

   a. In the View List under **Media**, click **Patch Design**.

   b. In the **Patch Design** explorer, select the patch configuration that you want to configure.

   If you are creating a QuickPatch package:

   a. In the View List under **Patch Settings**, click **General Information**.

   b. In the **General Information** explorer, click **Build Settings**.

2. Click the **Advanced** tab, and then find the **Update Launcher Settings** area.

3. In the following settings, enter the values that you want to use for the properties of your **Update.exe** file.

   - Company Name
   - Product Name
   - Product Version
   - Description
   - Copyright

Tip *• The Product Version setting updates the file version and the product version for* **Update.exe***. Note that the file version always contains four fields. If you specify fewer than four fields, the remaining fields are filled with zeros. For example, if you specify a product version of* ***1.1****, the file version that is used in the version resources of* **Update.exe** *is* ***1.1.0.0****.*

If you leave the update launcher settings blank, InstallShield uses the default InstallShield values.

Specifying the Icon for the Update Launcher

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

InstallShield lets you specify the icon that should be used for your **Update.exe** update launcher. The icon can be in an .exe, .dll, or .ico file.

End users can see this icon when they view your **Update.exe** file in Windows Explorer. The icon is also displayed on the Properties dialog box for **Update.exe**; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties.

If you do not specify an icon, InstallShield uses a default icon for your **Update.exe** file.

Task ***To specify the icon for your Update.exe file:***

1. The procedure depends on whether you are creating a standard patch or a QuickPatch package.

   If you are creating a standard patch in a Basic MSI or InstallScript MSI project:

   a. In the View List under **Media**, click **Patch Design**.

   b. In the **Patch Design** explorer, select the patch configuration that you want to configure.

   If you are creating a QuickPatch package:

   a. In the View List under **Patch Settings**, click **General Information**.

   b. In the **General Information** explorer, click **Build Settings**.

2. Click the **Advanced** tab, and then find the **Update Launcher Settings** area.

3. In the **Icon** setting, specify the fully qualified name of the file that contains the icon that InstallShield should use when it creates the **Update.exe** file at build time.

   To specify a file, type an explicit path or path variable, or click the ellipsis button (**...**) to open the **Change Icon** dialog box, in which you can click the **Browse** button to select a file.

   By default, the icon with index 0 is used; to specify a different icon, either select an icon in the **Change Icon** dialog box or append the icon's index or resource ID (preceded by a minus sign) to the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100.

The next time that you build the release, InstallShield uses the icon that you specified for your **Update.exe** file.

Patch Sequencing

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

InstallShield enables you to specify the order that Windows Installer version 3.0 or later should apply small-update patches to an installed product, regardless of the order in which they are provided to the target machine. With patch sequencing data, you can ensure that Windows Installer knows the intended relationships among the upgrades packaged within a family of patches. Consequently, applying patch 1 of a product after patch 2 has already been applied will register patch 1 without overwriting patch 2 files. For versions of Windows Installer earlier than version 3.0, the patch sequence is ignored, and any small-update patches are applied to the product in the order that they are provided to the target machine.

The patch sequencing functionality available with Windows Installer 3.0 and later simplifies the patch creation process. The following sections show how.

Creating Patches to Be Applied with Versions of Windows Installer Earlier than 3.0

If you need to create your patches so that they can be applied to your product via versions of Windows Installer earlier than 3.0, it is recommended that you avoid creating small updates. Small updates do not change the product version; therefore, external programs, including installers for later versions of your product, cannot distinguish a product with the small update applied from one without the small update. For scenarios limited to versions of Windows Installer earlier than 3.0, you need to plan around such limitations of the installer, thus targeting an increasing number of possible earlier product states. The sample application lifecycle presented in the following table illustrates the resulting complexity.

Table 8-5 • Sample Application Lifecycle for Patches Applied with Versions of Windows Installer Earlier than 3.0

Application Package	Product Version	Previous Setups Targeted by Package
1. Base installation	1.0	—
2. Minor upgrade	1.1	1.0
3. Minor upgrade	1.2	1.0, 1.1
4. Minor upgrade	1.3	1.0, 1.1, 1.2
5. Minor upgrade	1.4	1.0, 1.1, 1.2, 1.3
6. Major upgrade	2.0	1.0, 1.1, 1.2, 1.3, 1.4

Creating Patches to Be Applied with Windows Installer 3.0

With the patch sequencing functionality available with Windows Installer 3.0 and later, you can safely use a small-update patch to deliver a discrete upgrade for several different versions of a product, even though small updates do not change the product version. Unlike a small update, a minor upgrade changes the product version. Minor upgrades also form the framework for the sequencing of small-updates patches. If a small update for version 1.1 of a product is applied to version 1.2, the installer registers the small update on the target system and applies it as if it were encountered before the 1.2 minor upgrade was applied.

Small-update patches also enable Windows Installer 3.0 and later to maintain a valid product state as other patches are applied and removed individually from the product. In addition, patch sequencing lets you generate upgrade packages from a smaller set of earlier product states without requiring you to consider every possible combination of patches that could exist on the target machine. The sample application lifecycle presented in the following table illustrates this advantage.

Table 8-6 • Sample Application Lifecycle for Patches Applied with Windows Installer 3.0

Application Package	Patch Sequence Number	Product Version	Previous Setups Targeted by Package
1. Base installation	—	1.0	—
2. Small update	1	1.0	1.0
3. Small update	2	1.0	1.0

Table 8-6 • Sample Application Lifecycle for Patches Applied with Windows Installer 3.0 (cont.)

Application Package	Patch Sequence Number	Product Version	Previous Setups Targeted by Package
4. Small update	3	1.0	1.0
5. Small update	4	1.0	1.0
6. Minor upgrade	—	1.1	1.0

All of the small updates in the table above belong to the same patch family. Windows Installer 3.0 and later uses the patch family to compare a small-update patch with all of the other patches within the same family and determine the order in which each of the patches should be applied to the target machine. Patch sequences are added to the **MsiPatchSequence** table of the patch package database. This table defines the relationships between patches that target the same family of patches.

Creating Patches to Be Applied with Windows Installer 3.1

The patch sequencing functionality available with Windows Installer 3.1 enables you to further simplify the patch creation process with respect to minor upgrades. The **Minor Update to Target RTM Version (MSI 3.1 Required)** property on the Advanced tab of the Patch Design view lets you specify whether you want a minor-upgrade patch to target the release to manufacturing (RTM) version of the product (or the most recent major upgrade of the product, if one has been installed). You have two options for this property:

- If you want your minor-upgrade patch to essentially remove all of the patches up to the RTM version of the product (or the most recent major upgrade of the product, if one has been installed) before applying the current minor-upgrade patch, select Yes for this property. With this option, all patches (with or without sequencing data) are removed. You do not need to target additional baseline versions and thus increase the patch payload. All end users can successfully apply the patch without applying any intermediate patches.
- If you do not want your minor-upgrade patch to remove all of those earlier patches, select No. If you select this option, it may be necessary for your patch to contain the information needed to target each of the earlier minor upgrades that were created after the RTM (or the most recent major upgrade of the product, if one was created).

  For example, if you are creating a minor-upgrade patch for service pack 2 and you select No for this property, your patch needs to target the minor-upgrade patch for service pack 1. You could also optionally target other baselines (such as RTM); doing so would increase the patch payload. Note that if you do not target the RTM version, any end user who has the RTM version but not the service pack 1 minor-upgrade patch would need to install service pack 1 before service pack 2.

Creating Patches to Be Applied with Windows Installer 4.5

If a superseded patch installs a component for the product but later a superseding patch removes that component, the component's feature state may be changed to advertised, and it may not reinstalled. In addition, none of the remaining components associated with that feature can be maintained. This can be a problem on target systems that have Windows Installer 4.0 or earlier. The patch sequencing functionality available with Windows Installer 4.5 offers enhanced supersedence robustness to avoid this issue.

You can specify that a component in the current patch should be flagged for uninstallation in order to avoid leaving this component orphaned on the target system after a future superseding patch is applied. If a subsequent patch is installed and it is flagged to supersede the first patch, Windows Installer 4.5 can unregister and uninstall this component if appropriate.

Windows Installer 4.5 supports two methods for flagging superfluous components for uninstallation if they are superseded:

- Select Yes for the Uninstall Superseded Component setting for an individual component to indicate that you want it to be uninstalled if appropriate. If you select Yes, InstallShield adds the **msidbComponentAttributesUninstallOnSupersedence** attribute to the component in the **Component** table.

  To access the Uninstall Superseded Component setting, open the Components view and then select the component that you want to configure. The setting is displayed in the grid on the right. The default value of this setting is No.

- To indicate that you want all superfluous components in a superseded patch to be uninstalled, set the value of the Windows Installer property **MSIUNINSTALLSUPERSEDEDCOMPONENTS** to 1. You can set the value of the property from the command line or by adding this property to your project through the Property Manager view.

Windows Installer 4.0 and earlier ignore the **msidbComponentAttributesUninstallOnSupersedence** attribute and the **MSIUNINSTALLSUPERSEDEDCOMPONENTS** property.

Patch Uninstallation



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI—if the InstallScript user interface (UI) style is the new style (which uses the InstallScript engine as an embedded UI handler)*
- *QuickPatch*

This information does not apply to InstallScript MSI projects in which the InstallScript UI style is the traditional style (which uses the InstallScript engine as an external UI handler). To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

Windows Installer 3.0 or later supports the uninstallation of patches for small updates and minor upgrades; the patches to be uninstalled must have been installed with Windows Installer 3.0 or later. When a patch is uninstalled, the product is returned to the state it was in before the patch was uninstalled. With earlier versions of Windows Installer, an end user who wants to remove a patch needs to uninstall the patched product and then reinstall the product without applying the patch. In this case, any patches that should not be removed must be reapplied.

End users can uninstall patches through Add or Remove Programs on systems running Windows XP SP2 and later. For systems running Windows Installer 3.0 or later with earlier versions of Windows, patches must be uninstalled from the command line. For more information, see Uninstalling Patches in the Windows Installer Help Library.

The uninstallation of patches is not enabled by default because not all patches can be uninstalled. Following are examples of scenarios in which a patch cannot be uninstalled, or when certain functionality does not work during patch uninstallation:

- A major upgrade packaged as a patch cannot be uninstalled.

- If a patch adds a row to certain Windows Installer tables, the patch cannot be uninstalled. If your patch adds a row to any of these tables, one of the patch validators (Val0015) warns you that the patch will not be uninstallable. For a list of the applicable tables, see Val0015.
- It is possible that an end user may not be able to remove a patch that adds content to the **RemoveFile** or **RemoveRegistry** tables. If the patch is designed to remove a file or registry entry that was not included in the original installation package, uninstalling the patch will not restore that file or registry entry.
- If you create an uninstallable patch that contains SQL script support that was added through the SQL Scripts view, the SQL script will not run during the patch uninstallation.

It is recommended that you thoroughly test the uninstallation of your patch before deploying it to your end users. To learn more about patches that are not uninstallable, see Uninstallable Patches in the Windows Installer Help Library.

Non-Administrator Patches



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Windows Installer 3.0 and later enables you to create patches that can be installed by non-administrators. Non-administrator patches can be used if all of the following criteria are met:

- The target machine must be running Windows Installer 3.0 or later on the Microsoft Windows XP or later client platform. Server platforms are not supported.
- The application was installed from a removable media such as a CD-ROM or DVD.
- The application was installed in a per-machine context.



Note • *If the ALLUSERS property is overwritten at the command line, non-administrator patches will fail.*

- The base installation must include the certificate that will be used to sign all subsequent patches.
- The base installation must include the **MsiPatchCertificate** table. This table provides the signer certificate that will be used to verify the digital signature of subsequent patches when they are applied by a non-administrator. If necessary, this table can contain multiple certificates, and subsequent patches would need to be able to verify at least one of the certificates. For more information, see Preparing Installations for Non-Administrator Patches.
- The non-administrator patch must contain the **MsiDigitalCertificate** table. This table contains the signing certificates for the signed patches. For more information, see Signing a Patch Package or Signing a QuickPatch Package.

If any of the above criteria are not met, end users cannot install the digitally signed patch in a locked-down environment.

A typical scenario in which non-administrator patches are used is the computer game industry. Some computer game users are children who might not have access to areas of the system other than folders in their own user profile and registry keys under HKEY_CURRENT_USER. Their parents would have administrative access to the

machines so that they can control what is installed and what their children can access. Parents would install any and all applications. If patches are available for the installed software, children would be able to download and install non-administrator patches without help from their parents, as long as all of the above criteria have been met.

Creating and Applying Patches



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

This section of the help discusses how to create and apply patches to an installation. It also covers how to enable the uninstallation of a patch, use patch sequencing, and sign a patch package.

Adding a New Patch Configuration Item



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*



Task ***To add a new patch configuration item:***

1. In the View List under **Media**, click **Patch Design**.
2. Right-click **Patch Design** and click **Add New Patch Configuration**.

InstallShield adds a new patch configuration item.

Adding a New Latest Setup Item



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*



Task ***To add a new latest setup item:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, right-click the appropriate patch configuration item and click **Add New Latest Setup**.

InstallShield adds a new latest setup item.

Adding a New Previous Setup Item

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To add a new previous setup item:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, right-click an existing previous setup item and click **Add New Previous Setup**.

InstallShield adds a new previous setup item.

Adding an External File

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To add an external file:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, right-click **Additional External Files** and click **Add New External File**.

InstallShield adds a new external file item. You can configure this file in the pane on the right.

Configuring Patch Settings

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The best way to create and configure a patch is to use the Patch Design view. Before you work in the Patch Design view, you should have already built the projects for your earlier and later product versions.

Each patch designed in the Patch Design view begins as a patch configuration. A patch configuration consists of one latest setup item and one or more previous setup items. At the patch configuration level, you can configure properties on the following property tabs:

- Common

- Identification
- Digital Signature
- Sequence
- Advanced

After you have set the patch configuration properties, drill down the hierarchy in the explorer, and configure the latest and previous setup items.

Specifying Identification Information for a Patch



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Windows Installer 3.0 and later adds an Add or Remove Programs entry for each patch that is applied to a target system, even if the patch is not uninstallable.



***Tip** • Every time that you change the latest setup for a patch configuration in the Patch Design view, InstallShield uses the Add or Remove Programs information from the latest setup as the values for the Identification tab settings. You can override the values on the Identification tab as needed.*



Task ***To specify identification information for a patch:***

1. In the View List under **Media**, click **Patch Design**.
2. Create a patch configuration if you have not already done so.
3. In the **Patch Design** explorer, click the patch configuration.
4. Click the **Identification** tab.
5. Configure each of the settings.

Enabling the Uninstallation of a Patch



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI—if the InstallScript user interface (UI) style is the new style (which uses the InstallScript engine as an embedded UI handler)*

This information does not apply to InstallScript MSI projects in which the InstallScript UI style is the traditional style (which uses the InstallScript engine as an external UI handler). To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

Windows Installer 3.0 and later supports the uninstallation of patches for small updates and minor upgrades. However, not all patches can be uninstalled. For details about the uninstallation of patches, including the associated limitations, see Patch Uninstallation.

Task ***To enable the uninstallation of a patch:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, click the patch configuration that you want to configure.
3. Click the **Common** tab.
4. Select the **Allow Patch to be Uninstalled (Requires Windows Installer 3.0)** check box.

Sequencing Patches

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Before you define a sequence for patches, you should create the necessary upgrade items in the Upgrades view. Then you should add a patch configuration item in the Patch Design view. Once you have completed those tasks, you can specify the sequence for the patches.

Task ***To define a patch sequence:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, click the patch configuration that you want to configure.
3. Click the **Sequence** tab.
4. Do one of the following:
   - To use default patch sequencing that InstallShield generates, select the Use Default Patch Sequencing check box.
   - To use no patch sequencing, clear the Use Default Patch Sequencing check box.
   - To specify your own custom sequencing:
     a. Clear the **Use Default Patch Sequencing** check box.
     b. Click **Add**. The **Patch Sequence** dialog box opens.
     c. Define the sequence information as needed.

Signing a Patch Package

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Windows Installer 3.0 and later enables you to create patches that can be installed by non-administrators. Non-administrator patches can be used if strict criteria are met. For example, the base installation that the patch will update must include the certificate that will be used to sign the patch package. To learn about the other criteria that must be met, see Non-Administrator Patches.

Note *• If you want to digitally sign the files—such as your application's executable fils—in your patch, you can specify which files should be signed through the Signing tab in the Releases view.*

Task ***To sign a patch package:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, click the patch configuration that you want to configure.
3. Click the **Digital Signature** tab.
4. Select the **Sign the Patch Package** check box.
5. Configure the digital signature settings.

Password-Protecting a Patch Package

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For added security, you can password-protect your patch. When you password-protect your patch, any end user who wants to apply your patch must enter a case-sensitive password to launch your update.

Task ***To password-protect a patch package:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, click the patch configuration that you want to configure.
3. Click the **Advanced** tab.
4. For the **Password Protect Launcher** setting, select **Yes**.
5. For the **Launcher Password** setting, specify the password that you want your patch to use.

***Note** • The password is case-sensitive.*

Specifying Whether to Build an Update.exe Update Launcher for a Patch

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

InstallShield lets you specify whether you want InstallShield to create an **Update.exe** update launcher for your patch package.

An **Update.exe** update launcher is required if you want to automatically update or install the Windows Installer engine on a target system if necessary. For more information on when an **Update.exe** launcher is required, see Patching Considerations.

If you configure InstallShield so that it does not create an **Update.exe** update launcher, it creates an .msp file.

Task ***To specify whether to include an Update.exe update launcher for a patch:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, click the patch configuration that you want to configure.
3. Click the **Common** tab.
4. To include **Update.exe**, select the **Create Update.exe** check box.

   To avoid including **Update.exe**, clear the **Create Update.exe** check box.

***Tip** • You can also use the Advanced tab to specify whether you want to include* **Update.exe**.

Copying a Latest Setup Item to Another Patch Configuration

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

If you have added more than one patch configuration item to the Patch Design view, you can copy the latest setup item within one patch configuration to another patch configuration.

Task ***To copy a latest setup item from one patch configuration to another:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, drag the latest setup item that you want to copy to the patch configuration item that does not have it.

InstallShield adds the latest setup item, including all of its previous setup items and additional external files, to the patch configuration.

Removing Patch Configuration Items

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To remove a patch configuration item:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, right-click the patch configuration that you want to delete and then click **Delete**.

Removing Latest Setup Items

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To remove a latest setup item:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, right-click the latest setup item that you want to delete and then click **Delete**.

Removing a Previous Setup Item

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Task ***To remove a previous setup item:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, right-click the previous setup item that you want to delete and then click **Delete**.

Building a Patch

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Once you have configured the settings for a patch configuration in the Patch Design view and created a new release for the patch in the Releases view, you are ready to build a patch for your upgrade.

Task ***To build a patch:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, right-click the patch configuration item for which you want to build a patch, and then click **Build Patch**.

InstallShield builds the patch according to the settings you specified for the patch configuration item. The patch is built in the location that you specified on the Common tab for the patch configuration; InstallShield appends the following subdirectories to the output location that you specified:

*****`PatchConfigName`**\Patch**

By default, InstallShield also performs upgrade and patch validation every time that you build a patch. For more information, see Validating Upgrades, Patches, and QuickPatch Packages.

The build will return an error and exit if your latest setup version was built compressed.

Applying Patches

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

A patch package (.msp) file contains the transforms and instructions necessary for upgrading one or more installed versions of a product.

Before you can apply a patch package, Windows Installer version 1.1 or later must be present on the system. In addition, the package that you want to update must be installed locally or as an administrative image. A further requirement is that the existing installation package be installed with the same privileges and for the same users as the patch package. For example, if the product was installed for all users, the update should be installed for all users, as well.

The easiest way to apply a patch is to double-click the .msp file in Windows Explorer or right-click the file and then click Open. When you apply a patch for a minor upgrade, a PatchWelcome dialog is the first dialog that opens. When you apply a patch for a major upgrade, the full dialog sequence appears, as when you run an installation standalone.

From the command line, you can apply a patch with the MsiExec.exe `/p` option. Type the following statement to apply a patch package located at **X:\Product Updates\Build 36\PatchForV1.msp**:

```
msiexec /p "X:\Product Updates\Build 36\PatchForV1.msp"Update.exe
```

If you selected the Create Update.exe check box for your patch configuration in the Patch Design view, InstallShield wraps your .msp file in an executable file. **Update.exe** launches your patch package with the following command-line expression:

```
msiexec /p <path to .msp file> REINSTALL=ALL REINSTALLMODE=omus
```

Applying Patches in Silent Mode

There are two ways you can apply a patch in silent mode: either launch MsiExec.exe with the `/qn` command-line parameter, or pass `/s` to **Update.exe**.

There is an important consideration to bear in mind when applying a patch in silent mode. In order to operate correctly, the Windows Installer property **REINSTALL** must be set to **ALL** and **REINSTALLMODE** must be set to **omus** whenever you apply a patch. Since **Update.exe** always sets these properties at the command line, you do not have to do anything extra if your patch package is applied with **Update.exe**.

When a patch package is applied with a full user interface, one of your installation's default dialogs, PatchWelcome, is displayed. It includes control events to set **REINSTALL** and **REINSTALLMODE** with the correct options. However, since this dialog is not displayed when the end-user interface is suppressed, you must set the properties at the command line, as demonstrated below:

```
msiexec /p <path to .msp file> /qn REINSTALL=ALL REINSTALLMODE=omus
```

Because a patch does not modify the existing cached .msi database, including the **v** setting for **REINSTALLMODE** is unnecessary.

Creating a QuickPatch Project



Project • *You can create a QuickPatch to update an earlier version of your product only if the earlier version was installed using Windows Installer. The installation for the earlier version of your product must have been created with one of the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

A QuickPatch project is recommended for installation authors who want to ship small, single upgrades to their users. QuickPatch authoring provides a simple alternative to creating a patch in the Patch Design view, even though it provides less customization. Fundamentally, both patch creation methods produce the same deliverable types: .msp and .exe files.

To create a QuickPatch project for an existing .msi file or an already existing QuickPatch, use the Create New QuickPatch Wizard.

Creating a QuickPatch Project for an Existing QuickPatch



Project • *This information applies to QuickPatch projects.*

You can create a new QuickPatch project based on an existing QuickPatch project. This creates a patch that you can deliver to any end users who need to apply a patch for the original application or specific patched versions of the application.



Caution • *If you open and modify any of the releases listed in the History area of the General Information view, your latest project will not work, since intermediate data shared across the releases in the History may have been lost or altered. Therefore, whenever you create a QuickPatch project for an existing QuickPatch project, InstallShield changes the existing QuickPatch project (.ism file) to read-only mode.*



Task ***To create a QuickPatch project for an existing QuickPatch project:***

1. In InstallShield, open the QuickPatch project (.ism file) that you would like to patch.
2. Open the **General Information** view.
3. In the **General Information** explorer, click **History**.
4. Click **Create Next Version**.

Your new QuickPatch project opens in InstallShield.

You can also use the Create New QuickPatch Wizard to create a QuickPatch for an existing QuickPatch.

Specifying Whether to Streamline the QuickPatch Package



Project • *This information applies to QuickPatch projects.*

When you are configuring a QuickPatch project, you have the ability to specify whether you want InstallShield to streamline the creation of your QuickPatch package to build as simple a package as possible. The goal of QuickPatch streamlining is to generate a QuickPatch package that has fewer new subfeatures and custom actions than a non-streamlined QuickPatch package.

For example, if your QuickPatch project includes a new file or registry entry and InstallShield does not use QuickPatch streamlining, InstallShield creates a new subfeature for that file or registry entry. InstallShield also adds one or more prebuilt InstallShield custom actions to work around certain Windows Installer patch requirements. However, if InstallShield does use QuickPatch streamlining, the file or registry entry is added to an existing feature, and no special prebuilt InstallShield custom actions are required.



Note • *InstallShield cannot streamline the creation of a QuickPatch package in the following scenarios:*

- *The QuickPatch package removes an installed file.*
- *The QuickPatch package removes or renames a registry key.*
- *The QuickPatch package targets a non-streamlined QuickPatch image. That is, you cannot use QuickPatch streamlining if you select the check box in the History area of the General Information view for a QuickPatch that did not use QuickPatch streamlining. If you try to build a streamlined QuickPatch that targets one or more non-streamlined QuickPatch images, InstallShield displays a build warning, and it does not use streamlining.*

Also note that you can apply no more than 15 cumulative, non-streamlined QuickPatch packages to a base .msi package or major upgrade package. If you exceed the limit, error 2701 occurs while you are applying the patch. The exact limit is determined by the depth of the package's feature tree and the subfeature that each non-streamlined QuickPatch adds to that tree. Windows Installer limits the depth of the feature tree to 16 levels.

A streamlined QuickPatch package does not contain any new subfeatures, so it does not have this limitation.



Task ***To specify whether to streamline the QuickPatch package:***

1. In the View List under **Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Build Settings**.
3. Click the **Advanced** tab.
4. In the **Streamline QuickPatch** setting, select the appropriate option:
   - To streamline the QuickPatch package, select **Yes**. This is the default value for new QuickPatch projects.
   - To avoid streamlining the QuickPatch package, select **No**.

Specifying Target Releases for Patching



Project • *This information applies to QuickPatch projects.*



Task ***To specify which releases should be patched by your QuickPatch project:***

1. In the View List under **Define Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **History**.
3. Select or clear the check boxes next to the intermediate releases—the releases in between the Base QuickPatch Image and the current QuickPatch image—to specify whether or not you want them to be patched by the current project.

***Note** • You cannot clear the check box for the Base QuickPatch Image or the current project. If you clear the check box of an intermediate QuickPatch project, your current QuickPatch project cannot be used to upgrade a machine that has that intermediate image as the latest image of your application.*

For example, suppose you distributed version 1.0 as an installation and later released a patch that upgrades the original installation to version 1.1. If you create a version 1.2 QuickPatch and the check box for the 1.1 QuickPatch is not selected in the History, your 1.2 QuickPatch can be used to upgrade the 1.0 release but not the 1.1 release.

Specifying Custom Actions for the QuickPatch to Execute

***Project** • This information applies to QuickPatch projects.*

Task ***To specify which custom actions should be run when your QuickPatch is applied:***

1. In the View List under **Define Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Custom Action**.
3. Select or clear the check boxes next to the custom actions to specify whether or not you want them to be run during the installation of your QuickPatch.

Adding Files to a QuickPatch

***Project** • This information applies to QuickPatch projects.*

Task ***To add a file to your QuickPatch:***

***Note** • QuickPatch projects do not have support for defining a new target destination for new files that you are adding. Therefore, when you add a new file in a QuickPatch project, the destination location must be a folder that was defined in the original installation.*

1. In the View List under **Define Patch Settings**, click **Files**.
2. Right-click the **Files To Patch** explorer and then click **Insert New File**. The **Open** dialog box opens.
3. Click the file that you want to add, and then click **Open**. InstallShield adds the file to the **Files To Patch** explorer.
4. Click the new file and then configure its settings.

Specifying Identification Information for a QuickPatch

Project • *This information applies to QuickPatch projects.*

Windows Installer 3.0 and later adds an Add or Remove Programs entry for each QuickPatch package that is applied to a target system, even if the QuickPatch is not uninstallable.

Task ***To specify identification information for a QuickPatch:***

1. In the View List under **Define Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Build Settings**.
3. Click the **Identification** tab.
4. Configure each of the settings.

Enabling the Uninstallation of a QuickPatch

Project • *This information applies to QuickPatch projects.*

Windows Installer 3.0 and later supports the uninstallation of QuickPatch packages for small updates and minor upgrades. However, not all QuickPatch packages can be uninstalled. For details about the uninstallation of standard patches and QuickPatch packages, including the associated limitations, see Patch Uninstallation.

Task ***To enable the uninstallation of a QuickPatch package:***

1. In the View List under **Define Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Build Settings**.
3. Click the **Common** tab.
4. Select the **Allow Patch to be Uninstalled (Requires Windows Installer 3.0)** check box.

Sequencing QuickPatch Packages

Project • *This information applies to QuickPatch projects.*

Task ***To define a sequence for a QuickPatch package:***

1. In the View List under **Define Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Build Settings**.

3. Click the **Advanced** tab.
4. In the **Create patch sequencing entry** setting, select the appropriate option:
   - To use default patch sequencing that InstallShield generates, select **Yes**.
   - To use no patch sequencing, select **No**.

Signing a QuickPatch Package



Project • *This information applies to QuickPatch projects.*

Windows Installer 3.0 and later enables you to create patches that can be installed by non-administrators. Non-administrator QuickPatch packages can be used if strict criteria are met. For example, the base installation that the patch will update must include the certificate that will be used to sign the patch package. To learn about the other criteria that must be met, see Non-Administrator Patches.



Note • *With QuickPatch projects, you can digitally sign the patch package and the* **Update.exe** *file. If you want to digitally sign individual files—such as your application's executable files—in your QuickPatch package, you must manually sign them and then add them to your project. You can use* **SignTool.exe***, which is included with the Windows SDK, to manually sign your files.*



Task ***To sign a patch package:***

1. In the View List under **Define Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Build Settings**.
3. Click the **Digital Signature** tab.
4. Select the **Sign the Patch Package** check box.
5. Configure the digital signature settings.

Password-Protecting a QuickPatch Package



Project • *This information applies to QuickPatch projects.*

For added security, you can password-protect your QuickPatch package. When you password-protect your QuickPatch package, any end user who wants to apply your QuickPatch must enter a case-sensitive password to launch your update.

Task ***To password-protect a QuickPatch package:***

1. In the View List under **Define Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Build Settings**.
3. Click the **Advanced** tab.
4. In the **Password Protect Launcher** setting, select **Yes**.
5. In the **Launcher Password** setting, specify the password that you want your patch to use.

Note • *The password is case-sensitive.*

Specifying Whether to Build an Update.exe Update Launcher for a QuickPatch Package

Project • *This information applies to QuickPatch projects.*

InstallShield lets you specify whether you want InstallShield to create an **Update.exe** update launcher for your QuickPatch package.

An **Update.exe** update launcher is required if you want to automatically update or install the Windows Installer engine on a target system if necessary. For more information on when an **Update.exe** launcher is required, see Patching Considerations.

If you configure InstallShield so that it does not create an **Update.exe** update launcher, it creates an .msp file.

Task ***To specify whether to include an Update.exe update launcher for a QuickPatch:***

1. In the View List under **Define Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Build Settings**.
3. Click the **Common** tab.
4. To include **Update.exe**, select the **Create Update.exe** check box.

   To avoid including **Update.exe**, clear the **Create Update.exe** check box.

Tip • *You can also use the Advanced tab in the Build Settings area to specify whether you want to include* **Update.exe**.

Deleting Files from the Files To Patch Explorer



Project • *This information applies to QuickPatch projects.*



Task ***To delete a file from the Files To Patch explorer:***

1. In the View List under **Define Patch Settings**, click **Files**.
2. Right-click the file that you want to delete and then click **Delete**.

Modifying and Removing Installed Files with a QuickPatch



Project • *This information applies to QuickPatch projects.*



Task ***To modify or remove an installed file with your QuickPatch:***

1. In the View List under **Define Patch Settings**, click **Files**.
2. Right-click the **Files To Patch** explorer and then click **Patch Existing File**. The **Select File** dialog box opens.
3. Click the file that you would like to modify or delete. InstallShield adds the file to the **Files To Patch** explorer.
4. Click the file that you just added to the **Files To Patch** explorer and then configure its settings.



Tip • *As an alternative to steps 3 and 4 above, you can drag and drop files or folders from the* ***Original Setup Files*** *explorer to the* ***Files To Patch*** *explorer.*

Adding, Modifying, and Deleting Registry Data with a QuickPatch



Project • *This information applies to QuickPatch projects.*

When you add, modify, or delete registry data with a QuickPatch project, the procedures are basically the same as the ones that you perform for the original installation. The one difference is that before you can add registry data for a QuickPatch project, you must select an existing feature in the View Filter list at the top of the Registry view. All registry data must be associated with a feature that already exists in the original product because you cannot add new features or components with a QuickPatch.



Tip • *To change any registry settings that you modified for the QuickPatch project, you can right-click the item and then click Revert.*

Validating Upgrades, Patches, and QuickPatch Packages

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Validation helps identify some common problems that may be encountered when attempting to upgrade an existing Windows Installer–based installation.

Note • *You can perform upgrading and patching validation on uncompressed releases only.*

The validation engine for upgrading and patching currently consists of different validators that are designed to test for a specific condition and report failure as necessary. Upgrading and patching validation was designed to facilitate the detection of upgrade scenario issues and is distinct from Windows Installer validation, which is better suited for validating first-time installation issues.

Validators

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

The following table contains a list of validators for upgrading and patching validation.

Table 8-7 • Upgrade and Patch Validators

Validator Number	Description
Val0001	This validator verifies that when the upgrade is applied, the files that have been removed from the latest version of the installation will be uninstalled from the target machine.
Val0002	This validator detects cases where **RemoveFile** table entries are scheduled to run during installation only.
Val0003	This validator verifies that the proper settings exist so that an upgrade can start successfully.
Val0004	This validator verifies that when the upgrade is applied, all files that have changed in the latest installation will be installed properly on the target system.

Table 8-7 • Upgrade and Patch Validators (cont.)

Validator Number	Description
Val0005	This validator verifies that feature InstallLevel values are configured so that no feature will be ignored in an upgrade.
Val0006	This validator detects when a minor upgrade is being attempted, even though a major upgrade should be performed.
Val0007	This validator verifies that the name of the .msi package has not changed.
Val0008	This validator verifies the integrity of the ActionProperty and Version elements of all records in the **Upgrade** table.
Val0009	This validator verifies that when the installation is run as an upgrade, registry entries that have been deleted from an installation will be removed.
Val0011	This validator detects when schema incompatibility exists. Schema incompatibility will prevent the creation of a patch.
Val0012	This validator verifies that new features are properly added to the latest installation so that they will be installed when the upgrade is run.
Val0013	This validator verifies that the end user is informed of the intended functionality of the Remove attribute for a major upgrade item.
Val0014	This validator warns you that because of a specific entry in the **Upgrade** table, part of the existing product will be left on the target machine when the major upgrade is applied.
Val0015	This validator warns you that if you want to package your upgrade as a patch in certain scenarios, end users will not be able to uninstall your patch. This validator applies to small updates and minor upgrades.

Val0001



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message (Error)

```
The file [1] with a target of [2] appears to have been removed from the setup, but does not appear in the
    RemoveFile table. This file will not be removed from the target machine when an upgrade is run unless
    the RemoveFile table has been authored.
```

[1] is the name of the file that has been removed, while [2] is the target path of this file.

Description

When a small update or minor upgrade is applied, any file or files that have been removed from the latest setup will not automatically be uninstalled from the target machine because a minor upgrade will recache the local .msi database. This recached copy no longer contains any reference to the file, since it has been removed from the setup.

To remove this file when the upgrade is run, you must add entries to the **RemoveFile** table. When the upgrade is run, these **RemoveFile** table entries will be executed, and the file will be deleted.

Note *• This validator does not apply to major upgrades because the uninstall-and-then-reinstall nature of major upgrades does not lend itself to this type of consideration.*

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

Using the file name and file destination from the previous version setup package, create an entry in the **RemoveFile** table using the Direct Editor. The file name can contain wildcard characters.

Note *• When authoring the* ***RemoveFile*** *table, you must specify a component. This component's install state is used to determine if a file matching the RemoveFile signature should be removed or not. Be sure to specify a component that you are certain will reinstall when the upgrade is applied.*

Val0002

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message (Note)

```
The RemoveFile table entry [1] has an InstallMode of 1. In an upgrade scenario, if the component [2] does
    not have to reinstall, the files attributed to this table entry will not be removed. You can change
    the InstallMode value to 3 so that the files will at least get removed on uninstall, or make sure
    that this RemoveFile table entry is associated with a component that will get reinstalled.
```

[1] is the name of the entry in the first column of the **RemoveFile** table that is causing this message. [2] is the name of the component with which this **RemoveFile** table entry is associated.

Description

When you author an entry in the **RemoveFile** table, you must specify a value in the InstallMode column. If this value is set to 1, this tells the installer that the specified file should only be removed upon installation of the associated component. In an upgrade scenario, if the associated component does not need to be reinstalled, then the **RemoveFile** table entry will not be executed, and the file will not be removed from the target machine.

In a subsequent uninstallation, you will need to manually remove this file and the directory that contains it since the file still exists on the target machine.

Note • *This validator does not apply to major upgrades because the uninstall-and-then-reinstall nature of major upgrades does not lend itself to this type of consideration.*

To perform this validation test, the validation engine will need to compare the latest setup to a previous version of that setup.

Corrective Action

This validator is only a note. It is designed to make you aware of a potential issue with your setup when you are creating an upgrade. If you do not intend on performing an upgrade or you are certain that the component associated with this RemoveFile entry will be reinstalled in an upgrade, you can choose to disregard this note.

You can also choose to change the InstallMode value to 3. In this case, a file will not be removed during installation, but it will always be removed during an uninstallation.

Val0003

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Note)

```
This setup will perform a SMALL upgrade of the referenced previous setup.
```

Message 2(Note)

```
This setup will perform a MINIOR upgrade of the referenced previous setup.
```

Message 3(Note)

```
This setup will perform a MAJOR upgrade of the referenced previous setup.
```

Message 4(Error)

```
The package code for the latest setup does not differ from that of the previous version. In order to
    perform an upgrade, the package code MUST change.
```

Message 5(Note)

```
The product version [1] differs from the product version [2] but only past the third element. The Windows
    Installer does not detect version differences past the third version element.
```

[1] is the version number of the previous installation. [2] is the version number of the latest installation.

Message 6(Error)

```
The setup needs to perform a Major Upgrade of the previous setup specified. However, the Upgrade Code for
    the previous setup [1] is not in the upgrade table. An upgrade will not occur.
```

[1] is the upgrade code of the previous installation.

Message 7(Error)

```
The setup needs to perform a Major Upgrade of the previous setup specified. While upgrade table entries
    exist for the Upgrade Code, there is no upgrade table entry that matches the Product Version [1] or
    Product Language [2].
```

[1] is the version number, and [2] is the language ID of the installation.

Message 8(Error)

```
A previous package does not contain an UpgradeCode property. A major upgrade will fail to uninstall any
    previous versions that do not have an UpgradeCode.
```

Description

This validator determines the type of upgrade that is necessary to update an earlier installation. For example, if you are creating a minor upgrade but this validator states that your installation is configured to perform a major upgrade, consider changing your minor upgrade to a major upgrade.

This validator also checks if the package code has changed. This check is performed for all upgrade types. Changing the package code is a requirement any time that you decide to release an installation, whether or not you intend to upgrade it. If the package code has not changed, you will see the error described in message 4.

If the product version has changed but only past the third version item, the validator displays the note described in message 5. This note is designed to alert you that changes to the fourth item in a version resource will not appear as unique versions to the installer at run time.

Additionally, if a major upgrade is required, this validator will check if the appropriate entries exist in the latest installation to remove the previous installation from the target machine. If these entries do not exist, you will see message 6 or 7. Message 6 is displayed when there are no settings related to the earlier product. Message 7 is displayed when there is a setting related to the earlier product, but the signature of that setting does not patch the signature of the previous setup. Error 8 is displayed if there is a setting related to the earlier product, but the upgrade code was not found in the earlier installation.

Corrective Action

If you receive message 4, you must generate a new package code for the latest version of your installation. A good option is to set the Generate Package Code property for the product configuration to Yes.

If you receive message 5, you can choose to disregard it since it is for informational purposes. However, note that having indistinguishable versions can create difficulties if you attempt to apply a major upgrade at a later date.

If you receive message 6, you need to add a major upgrade item.

There are several possible reasons why you might receive message 7. The first possibility is that the product version does not fall within the range of versions specified as potential target installations. If the version of the previous installation is either the upper bound or the lower bound of the acceptable range of product versions to upgrade,

you may want to check the minimum and maximum inclusive settings for your major upgrade item. Additionally, the product language may not be listed in the list of supported languages for upgrading. You will also get error message 7 if you have defined a major upgrade item for the previous setup but have marked that item to **Detect Only**.

If you encounter error 8, remove the earlier package that does not have an upgrade code.

Val0004



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message (Error)

```
The key file [1] in component [2] is versioned in the previous package and has a lower version or is
    unversioned in the upgraded package (old file version [3], new file version [4]). This will prevent
    the component from reinstalling in an upgrade scenario.
```

[1] is the name of the key file , [2] is the name of the component that contains the key file, [3] is the version number of the file in the earlier package, and [4] is the version number of the file in the upgrade.

Description

This validator verifies that when the upgrade is applied, all files that have changed in the upgrade will be updated properly on the target system.

At run time during a minor upgrade or small update, Windows Installer does not reinstall a component if its key file would be downgraded in the process. Therefore, the key file in the upgrade needs to have a version number that is equal to or greater than that of the key file in the old package.

The exception is when the end user sets the **REINSTALLMODE** property to **a**, which forces the installation of all files, regardless of version.



Note *• This validator does not apply to major upgrades because the uninstall-and-then-reinstall nature of major upgrades does not lend itself to this type of consideration.*

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

To resolve the error, ensure that the version number of the key file in the upgrade has a version number that is equal to or greater than that of the key file in the old package. As an alternative, you can create a major upgrade.

Val0005



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Warning)

```
The default InstallLevel for feature [1] is zero. If this feature gets enabled on install, you must
    author similar logic to ensure that it is also enabled in maintenance mode, otherwise in an upgrade
    the feature will be ignored.
```

[1] is the name of the feature whose **InstallLevel** is the cause of concern.

Message 2(Warning)

```
A condition for feature [1] may possibly set the InstallLevel for this feature to zero at runtime. If
    this feature gets enabled on install, you must author similar logic to ensure that it is also enabled
    in maintenance mode, otherwise in an upgrade the feature will be ignored.
```

[1] is the name of the feature whose **InstallLevel** is the cause of concern.

Description

The significance of setting a feature **InstallLevel** to zero is that it will cause the feature to appear as disabled at run time, and the user will not be able to enable it. It is tempting to set this feature to zero by default and then enable it as needed through a feature condition.

In an upgrade, if this feature condition does not get evaluated so that it can set the install level to a value other than zero, the **InstallLevel** for the feature will remain at zero. Therefore, no changes made to the components in that feature will be installed.

Another variation of this issue is that the feature **InstallLevel** defaults to something other than zero, but there exists a feature condition that sets this feature **InstallLevel** to zero at run time. This issue raises problems for major upgrades, since the previous installation is already in the field.

Val0005 applies to all upgrade types: major, minor, and small.

To perform this validation test, the validation engine will need to examine only the latest version of your setup.

Corrective Action

Ensure that any processing that can effect the evaluation of these feature conditions is also performed when an upgrade is run. Remember that in an upgrade, the user interface being displayed to the end user may be different from the user interface displayed the first time the installation is run.

Making sure that your feature **InstallLevel** values do not default to zero is important so that a major upgrade will be able to safely install and uninstall files.

Val0006



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Error)

```
The Component [1] identified by ComponentID [2] is missing from the newest version of your setup. You
    cannot delete components and still do a minor/small upgrade. You must perform a major upgrade.
```

[1] is the name of the component that has been removed. [2] is the ComponentID associated with the component that was deleted.

Message 2(Error)

```
The Feature [1] is missing from the newest version of your setup. You cannot delete features and still
    do a minor/small upgrade. You must perform a major upgrade.
```

[1] is the name of the feature that has been removed.

Description

If any components or features have been deleted from the latest version of the installation, you must perform a major upgrade to avoid potential problems associated with leaving resources and registrations on a machine.



Note *• If you move a feature or component to another location in the installation, you have effectively deleted it from its original location.*

This validator does not apply to major upgrades. It identifies when a minor upgrade or small update should be a major upgrade.

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

If you need to make major architecture changes to your installation, you should apply a major upgrade.

You can also use the RemoveFile, RemoveRegistry, and RemoveIniFile tables to clean up some resources that may be orphaned. However, this will not clean up the Windows Installer registration information for those components and features. Leaving that registration information behind can cause unpredictable results in a future upgrade installation or uninstallation of your product.

Val0007



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Error)

```
The MSI package name for the most recent setup [1] differs from the MSI package name of the previous
    setup [2]. Small/Minor upgrades require that the package name remain the same.
```

[1] is the name of the new .msi package that has changed. [2] is the name of the original .msi package.

Description

When you perform a minor upgrade or a small update, the previous and latest versions of your installation must have the same .msi package name. Attempting to perform a minor upgrade or a small update when the .msi file name has changed can lead to Windows Installer run-time error 1316.

This validator does not apply to major upgrades.

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

This is a limitation of the Windows Installer service. The only recourse here is to give your .msi packages the same name. To configure the name of the .msi package, you can use the MSI Package File Name setting on the General tab for a product configuration in the Releases view.

Val0008



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Warning)

```
In the Upgrade table, the Action Property [1] is not listed as a member of SecureCustomProperties. The
    defined upgrade may not function properly.
```

[1] is the name of one of the action properties specified in the **Upgrade** table.

Message 2(Error)

```
In the Upgrade table, the Action Property [1] is not public. The defined upgrade will not function
    properly.
```

[1] is the name of one of the action properties specified in the **Upgrade** table.

Message 3(Error)

```
In the Upgrade table, the Action Property [1] is already defined in the property table. The defined
    upgrade will not function properly.
```

[1] is the name of one of the action properties specified in the **Upgrade** table.

Message 4(Error)

```
In the Upgrade table, the Action Property [1] is used more than once. This may cause undesirable results
    at runtime.
```

[1] is the name of one of the action properties specified in the **Upgrade** table.

Message 5(Error)

```
In the Upgrade table, there exists a record where the MaxVersion [1] is less than the MinVersion [2].
    This will cause unpredictable results a runtime.
```

[1] and [2] are version numbers specified in the **Upgrade** table.

Description

For all records in the **Upgrade** table:

- Ensure that the **ActionProperty** is public.
- Ensure that the **ActionProperty** is unique.
- Ensure that the **ActionProperty** is not predefined in the **Property** table.
- Ensure that the **ActionProperty** is in the **SecureCustomProperties** list.
- Ensure that **MinVersion** is less than or equal to **MaxVersion**.

This validator applies only to major upgrades.

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

To find where the value of the Action property is entered, open the Upgrades view. Select the major upgrade item, and then click the Advanced tab. The Detect Property value is used for the Action property.



Note *• Note the following:*

- *The property is public if it is in all uppercase letters.*
- *The property is unique if it is not shared with any other major upgrade items.*
- *If the property is defined in the* ***Property*** *table, go to the Property Manager and delete it.*
- *If the property is not on the SecureCustomProperties list, you can add this to the* ***SecureCustomProperties*** *property in the Property Manager. Multiple properties are delimited by semicolons.*
- *Find the major upgrade item in the Upgrades view where the minimum version is higher than the maximum version and correct it.*

Val0009

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Warning)

```
A registry entry has been removed from the component [1]. There exists a corresponding entry in the
    RemoveRegistry table, however, that entry is not associated with this component. This may result in
    the registry entry not being removed when an upgrade is run.
```

[1] is the name of the component from which a registry entry has been removed in the latest version of the setup.

Message 2(Error)

```
A registry entry has been removed from the component [1]. This key must be added to the RemoveRegistry
    table, otherwise it will be orphaned by an upgrade. [2].
```

[1] is the name of the component from which a registry entry has been removed in the latest version of the setup. [2] is the missing registry entry in the form <Root>|<Key>|<Value>.

Description

When a small update or minor upgrade is performed, any registry entries that have been removed from the latest installation will not automatically be uninstalled from the target machine. The reason is that a minor upgrade will recache the local .msi database. This recached copy no longer contains any reference to these registry entries, since it has been removed from the installation.

To remove this registry entry, add entries to the **RemoveRegsitry** table so that it will be removed when the upgrade is applied. When the upgrade is run, these **RemoveRegistry** table entries will be executed, and the registry entry will be deleted.

Note *• This validator does not apply to major upgrades because the uninstall-and-then-reinstall nature of major upgrades does not lend itself to this type of consideration.*

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

Using the root, key, and value of a registry entry from the previous version of the installation package, create an entry in the **RemoveRegistry** table through the Direct Editor. The Value field can be authored with the minus character (–). This will result in the removal of the entire registry key with all of its values and subkeys.

***Note** • When you add an entry to the **RemoveRegistry** table, you are required to specify a component. This component's install state is used to determine if a registry entry matching this **RemoveRegistry** signature should be removed. Ensure that you specify a component that you know will be reinstalled when the upgrade is applied.*

Val0011

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Error)

```
The data types for the Version column of the TypeLib table differs from that of the Upgraded image. This
    will cause a failure to create the patch package.
```

Message 2(Warning)

```
The target image was created with Windows Installer 1.2 or previous, and the Upgraded Image was created
    with Windows Installer 2.0 or later. Although validation did not detect any data type conflicts,
    creating patches for packages that span this schema inflection point can be problematic.
```

Description

With the release of Windows Installer 2.0, the schema of the .msi installation package changed. The data type of the **Version** column in the **TypeLib** table changed from I2 to I4. It is not possible to patch .msi installations using the old schema with .msi packages using the new schema.

***Note** • This validator does not apply to major upgrades because the uninstall-and-then-reinstall nature of major upgrades does not lend itself to this type of consideration.*

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

You can still distribute your upgrade as a minor upgrade, however, you will not be able to distribute your upgrade as a patch.

Val0012

***Project** • This information applies to the following project types:*

- *Basic MSI*

- *InstallScript MSI*
- *QuickPatch*

Message 1(Error)

```
The new feature [1] is defined as a root level feature. New features MUST be defined as sub-features of
    features that already exist in the original setup.
```

[1] is name of the feature causing the validation message.

Message 2(Error)

```
The validation engine has detected the new feature [1]. This feature must have the 'FavorParent' and
    'Required' attributes set to Yes in order to install in an upgrade scenario.
```

[1] is name of the feature causing the validation message.

Description

If you intend to distribute the latest version of your installation as a small update or a minor upgrade, you must follow certain rules if you decide to add features to the installation. The first rule is that the feature must be added as a child to an existing feature. The second rule is that the feature must have its Remote Installation property set to FavorParent and its Required property set to Yes.

If these two rules are not followed, the contents of the new feature will not be installed when the upgrade is applied. Instead, the end user will need to run the installation in maintenance mode and manually select the feature to be installed locally.

This validator does not apply to major upgrades.

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

Add the new feature as a child of an existing feature, and set its Remote Installation property set to FavorParent and its Required property set to Yes.

Val0013



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Warning)

```
Validation has detected that you are using the Remove column in the Upgrade table for the entry [1].
    Doing this will result in the incomplete uninstallation of the referenced setup. For the most common
    upgrade scenarios, you should not author the Remove column of the upgrade table.
```

[1] is the Upgrade code being referenced in the **Upgrade** table.

Description

When the remove attribute of a major upgrade item is left empty, it will remove all of the features from the target product, effectively uninstalling the entire product.

If you place one feature name in the remove attribute of a major upgrade item, only that feature will be removed, and all of the other features of the existing product will be left on the target machine along with the target product itself. This means that you will be left with two products on the target machine with two entries in Add or Remove Programs.

This feature is intended for users who want to leave the target product on end users' machine while removing only some of its features.

This validator applies only to major upgrades.

To perform this validation test, the validation engine needs to examine the latest installation only.

Corrective Action

This is a warning. If the behavior described above is what you desire, you can disregard this warning. Otherwise, do not use the Remove attribute of a major upgrade item.

Val0014



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Error)

```
The component identified by ID [1] has been removed from the latest version of the setup. On a Major
    Upgrade the resources contained within these components will get uninstalled, possibly deleting
    resources necessary for your application to run.
```

[1] is the component code that identifies a component in the earlier installation.

Description

This validator performs this check only if the major upgrade is configured to install new files, then remove unneeded files. The issue this validator is designed to protect you against does not exist if the major upgrade is configured to uninstall the product first, then reinstall it.

When the target upgrade mode is selected, the installer achieves this functionality by relying on reference count components on the target machine. The reference count for a required component is incremented when the latest application is installed, and it is decremented when the earlier application is removed. However, since the reference count never reaches zero, the persistent components remain on the target machine.

The problem exists when you delete a component from the installation, and then add the resources from that component to a different component. Instead, you should perform the following:

1. Install a new component with new resources.
2. Uninstall the old component with old resources.

When the old component is uninstalled in step 2, it may remove registry entries, .ini file changes, environment variables, and so on if they were added by the new component in step 1.

Since components are referenced by a GUID, and a new component GUID is generated every time you add a new component to your installation project, even a simple deletion or addition of a component that already exists can cause this behavior.

This validator applies only to major upgrades.

To perform this validation test, the validation engine compares the latest installation to an earlier version of that installation.

Corrective Action

To correct this problem, you can either configure your upgrade to uninstall the application first and then reinstall it, or you can add a component with a GUID equal to the one specified in this error message. Even though the component is empty, its existence helps Windows Installer keep accurate reference counts, and it helps to prevent components from being prematurely uninstalled.

Val0015



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

Message 1(Warning)

```
The [1] table contains new content. Therefore, if you are packaging this upgrade as a patch, you will not
    be able to make it an uninstallable patch.
```

[1] is the name of the table that is causing this message.

Description

When an end user uninstalls a patch, the product is returned to the state it was in before the patch was uninstalled. If a patch is not uninstallable, an end user who wants to remove the patch needs to uninstall the patched product and then reinstall the product without applying the patch.



Note • *If a target machine does not meet certain requirements (for example, it must have Windows Installer 3.0 or later), the patch will not be uninstallable, even if this validator warning is resolved. For more information about the requirements for uninstallable patches, see Patch Uninstallation.*

If a patch adds a row to any of the following specific tables, the patch cannot be uninstalled, even if the **Allow this patch to be uninstalled** check box is selected for the patch.

- BindImage
- Class
- Complus
- CreateFolder
- DuplicateFile
- Environment
- Extension
- Font
- IniFile
- ISLockPermissions
- IsolatedComponent
- ISSelfReg
- LockPermissions
- MIME
- MoveFile
- MsiServiceConfig
- MsiServiceConfigFailureActions
- MsiLockPermissionsEx
- ODBCAttribute
- ODBCDataSource
- ODBCDriver
- ODBCSourceAttribute
- ODBCTranslator
- ProgId
- PublishComponent
- RemoveIniFile
- SelfReg
- ServiceControl
- ServiceInstall
- TypeLib
- Verb

Note that under certain conditions, it is possible that an end user may not be able to remove a patch that adds content to the **RemoveFile** or **RemoveRegistry** tables. If the patch is designed to remove a file or registry entry that was not included in the original installation package, uninstalling the patch will not restore that file or registry entry.

This validator applies to small updates and minor upgrades only.

To perform this validation test, the validation engine compares the latest installation to an earlier version if the **Allow this patch to be uninstalled** check box is selected for the patch.

Corrective Action

This validator is designed to warn you that if you are creating a patch—or you later decide to package your upgrade as a patch—your end users will not be able to uninstall the patch unless you take corrective action to resolve it. To resolve this warning, do one of the following:

- Clear the **Allow this patch to be uninstalled** check box for the patch. If you do this, end users will not be able to uninstall the patch.
- Remove the new data from the table mentioned in the warning message. Note, however, that depending on your patch requirements, removing new table data may mean that your patch will not fix the issue in your application that led you to create the patch in the first place.

Patching Assemblies in the Global Assembly Cache



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *QuickPatch*

With Windows Installer 3.0 or later, the **MsiPatchOldAssemblyFile** and **MsiPatchOldAssemblyName** tables enable a patch package to patch an assembly in the global assembly cache (GAC) without making a run-time request for the original installation source. By default, InstallShield automatically generates entries for these tables when you build a patch or QuickPatch package. (To disable this automatic generation in a patch or QuickPatch project, set the **Generate MsiPatchOldAssembly tables** property to No. For a patch, this property is on the Advanced tab of the patch configuration item in the Patch Design view. In a QuickPatch project, this property is on the Advanced tab of the Build Settings item in the General Information view.)



Note *• To automatically generate entries for the* ***MsiPatchOldAssemblyFile*** *and* ***MsiPatchOldAssemblyName*** *tables, InstallShield must have write access to the latest version of the .msi package to which the patch applies. InstallShield modifies this package before creating the patch. InstallShield automatically attempts to make this package writable; if it cannot, a build warning is generated, and the table entries are not created.*

These table entries are applied only if the target system is running Windows Installer 3.0 or later. The patch runs if the system is running Windows Installer 2.0; however, these tables are ignored, and if the patch needs to update a file in the GAC, it makes a request for the original source package. InstallShield generates these table entries even if you include only the Windows Installer 2.0 engine in **Update.exe***, since a target system with Windows Installer 3.0 or later already installed can use them.*

Working with Differential and Full Releases

Project *• This information applies to InstallScript projects.*

InstallShield provides two different mechanisms for creating upgrades for InstallScript installations:

- You can package your upgrade as a full release that updates an existing product if an earlier version is installed, or behaves as a first-time installation if no earlier version is present.
- You can package your upgrade as a differential release that contains only the changed data (byte-level file differences) between the versions to be updated.

For step-by-step instructions on how to update an InstallScript installation by either creating a differential or full release, consult this section of the help.

Creating an InstallScript Release to Update Previous Versions

Project *• This information applies to InstallScript projects.*

InstallShield enables you to create a release for an InstallScript project to update one or more existing versions of your application. You can customize any necessary criteria through the script.

Note *• If the product that you want to update was not installed by an installation created with InstallShield X or later, InstallShield DevStudio, or InstallShield Professional 6.0 or later, you must create a new project. You may also want to consider creating a new project if you plan to include extensive changes in this update.*

Task ***To create an update release:***

1. Open a new or existing InstallScript project.
2. Verify that the product code matches the GUID with which the previous version of your product was installed. To check the product code, in the **General Information** view, inspect the value of the **Product Code** setting.

   Note *• If you are creating a new project and you want to update features that were installed by the original installation, ensure that each feature's GUID matches the GUID with which the previous version of that feature was installed. To check a feature's GUID, select it in the* ***Features*** *or* ***Setup Design*** *view and inspect the value of its GUID property.*
3. Specify a version number for your product update:
   c. In the View List under **Installation Information**, click **General Information**.
   d. In the **Version** setting, type the version number.
4. Indicate whether you want to build a differential or full release:
   a. In the **Releases** view, click the desired release.

b. If you want to build a differential release, set the value of the **Differential Media** property to **Yes** and continue with step 6. If you want to build a full release, leave the property set to **No** and continue with step 5.



***Note** • If you are using the **Release Wizard**, the **Update** panel is where you select the release format (Full or Differential).*

5. If you are building a full release, specify the versions of your product to which the update should be applied. In the **Supported Versions** property, type the applicable version numbers using a semicolon to separate the different versions (for example, **1.2.3;1.2.4**). If you leave the **Supported Versions** property blank or select the **Non-version specific**, the update is applied to all earlier versions of your product. Continue with step 7.



***Note** • These version numbers can also be specified in the Release Wizard by selecting version numbers from the box on the **Update** panel. Do not specify version numbers that correspond to releases that were created by versions of InstallShield Professional earlier than 6.0 because those releases cannot be updated.*

6. If you are building a differential release, you must use the **Release Wizard** to specify one or more existing releases to which the current project is compared when creating the new differential release.



***Note** • Do not specify existing releases that were created by versions of InstallShield Professional earlier than 6.0 because those releases cannot be updated. Only the specified existing releases can be updated by the resulting differential release. To review the criteria that determine whether a file in the current project is included in a differential release, see Differential vs. Full Releases.*

- To specify a release in the current project:

  a. In the **Releases** view, right-click the release name and select **Release Wizard**. The **Release Wizard** opens.

  b. In the **Update** panel, click **Add**. The **Add Existing Media** dialog box opens.

  c. Select the desired release.

  d. Click **OK**.

- To specify a release that is not in the current project:

  a. In the **Releases** view, right-click the release name and select **Release Wizard**. The **Release Wizard** opens.

  b. In the **Update** panel, click **Import**. The **Media File Properties** dialog box opens.

  c. Specify the Data1.hdr file of the desired release by either entering the fully qualified file name in the **Media Header File** box or select or clicking the browse button to browse for the file.

  d. Click **OK**.

- Confirm the specified release's version information is displayed in the **Differential Media** box of the **Update** panel. If no version information is displayed, do the following:

  a. Select the release in the list box and click **Modify**. The **Media File Properties** dialog box opens.

b. Select the **Specify the version information below** option and type or select a version number in the list.

c. Click **OK**.

7. Build the release.

See Migrating from InstallShield Professional 6.x for more information on using a version 6.x-created script with an update-enabled installation.

Notifying End Users about Upgrades Using FlexNet Connect



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

FlexNet Connect provides a mechanism that enables you to automatically notify your Web-connected end users when patches, updates, and product information for your application are ready for release.

FlexNet Connect Implementation

There are two cycles of tasks that you perform when using FlexNet Connect to automatically inform end users about updates: initial deployment and update deployment. Once you have performed the initial deployment steps for your application, each time you want to distribute an update of that application to end users, you perform the update deployment cycle of steps. To learn about the initial deployment steps, see Preparing Installations for Update Notifications.

Update Deployment

1. Use InstallShield to create an update for your application.

2. Register your new product version and product code with the FlexNet Connect Publisher site, a Web-based management portal.

3. Publish your update to the FlexNet Connect Publisher site, and set the **Message Status** to **Test**.

4. Test your update.

5. Publish your update to the FlexNet Connect Publisher site, and set the **Message Status** to **Active**.

FlexNet Connect includes a variety of options that can be purchased together for a complete solution, or separately for a customized solution. To learn more, visit the Flexera Software Web site.

9

Additional Installation Options

In addition to helping you create an installation package that installs products, InstallShield provides other installation options to help you enhance your final installation package. Some of these options include enabling multilingual installations, building conditional statements, and defining installation prerequisites.

Read more about additional installation options in the following sections.

Creating Multilingual Installations

Edition • *The Premier edition of InstallShield includes support for creating multilingual installations.*

InstallShield supports many features that enable you to customize your installation for global distribution. Using these features, you can create a single installation project that displays end-user text in multiple languages and can handle conditional installation of language-specific files.

Creating a multilingual installation primarily involves separating code from language-specific resources and files. You may also need to distribute separate files for your installation, such as graphics, license files, or custom actions, depending on the language in which the installation is running. Another consideration is whether you need to install different application files depending on the target system's locale.

Task

InstallShield enables the creation of multilingual installations by dividing installation authoring into the following distinct tasks:

- Specify the languages that your installation will include.
- Translate the strings for each supported language.
- Modify end-user dialogs as necessary for each language.
- Mark any language-dependent components.
- Select the languages to include in the release.

Globalization Tips

Consider these points when designing your product for a global audience:

- The goal of global distribution is a localized product that is international in scope and readily adaptable to specific areas of the world.
- The key to globalization is resource and code separation, plus country and language independence.
- Globalizing your installation requires a design that is simple and modular.
- Creating a worldwide specification package means incorporating global requirements into the installation specifications from the beginning.
- Make bitmaps and icons culturally sensitive. What may be acceptable in one country could be misleading or offensive in another.
- English strings are usually shorter than equivalent text strings in other languages. Translated strings grow an average of 30 to 40 percent. This implies that both static and temporary storage areas will increase in size.
- When designing prompts, use only one-half of the available space to allow for expansion.
- Avoid hard-coding element positioning and size on the screen, because these items can change when the element is translated.

Setting the Default Project Language

One of the project's supported languages must serve as its default language. The default language determines the following:

- The localizable strings that you specify throughout various views in InstallShield—such as the Display Name setting for a feature or the Description setting for a shortcut—are all from the default language's string entries, which you can see in the String Editor view. You can edit the values for the default language in the String Editor view, or in the other views throughout InstallShield.
- If your installation does not include the language selection dialog and it also does not include support for a target system's language, the default language is the language in which the installation runs on that target system. (The language selection dialog lets end users select which language version of the installation they would like to run.)

You can use the General Information view or the String Editor view to specify or change the default language.



Task ***To use the General Information view to specify or change a project's default language:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Default Language** setting, select the appropriate language.



Task ***To use the String Editor view to specify or change a project's default language:***

1. In the View List under **User Interface**, click **String Editor**.
2. In the **Default Language** box, select the appropriate language.



Tip *• You can override the default user interface language for a particular release. To do so, select the appropriate language in the Default Language setting on the Build tab in the Releases view.*

Settings for Languages



Edition *• The Premier edition of InstallShield includes support for creating multilingual installations.*

InstallShield lets you set languages at the project level, at the component level, and at the release level. Specifying languages at each level has different affects on your project. The following table describes various language-related settings.

Table 9-1 • Language Settings in InstallShield

Setting	Project Type	Description
General Information view setting—Setup Languages	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	 ***Project*** • *The behavior of this setting varies, depending on the project type.* In Basic MSI, InstallScript MSI, Merge Module, and Suite/Advanced UI projects, the Setup Languages setting lets you specify the languages that you want to be listed in the UI Languages setting in the Releases view. If a language is not listed for this Setup Languages setting at the project level, you cannot include that particular UI language in your project's releases. Advanced UI projects, which are available in the Professional edition of InstallShield, have support for only one language. Therefore, the Setup Languages setting in this project type is read-only. In InstallScript and InstallScript Object projects, the Setup Languages setting lets you specify the languages that you want to be listed in the Languages setting in the Components and Releases views in your project. In general, if a language is not listed for this setting at the project level, you cannot designate that a particular component in your project is targeted for that language; in addition, you cannot designate that the components and UI strings for a particular language are included in your project's releases. When you add a supported language to a Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, or Suite/ Advanced UI project through this setting, InstallShield adds string entries for that language to your project. The string entries include the built-in user-interface string resources that are already translated. In DIM projects, the Setup Languages setting lets you specify the languages that you want the project to support. When you add a language to your project through this setting, InstallShield adds string entries for that language to your project. The string entries include string resources that you would need to have translated. For more information, see Selecting the Installation Languages.

Table 9-1 • Language Settings in InstallShield (cont.)

Setting	Project Type	Description
Component setting—Languages	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The Languages setting for a component enables you to specify the languages for which the component is intended, for use in build-time filtering. By default, components are language independent, meaning that none of the component's data (such as files and registry entries) are specific to a particular language.  ***Project*** • *In InstallScript and InstallScript Object projects, if a language is not selected in the General Information view of a project, it is not listed as one of the available languages for the project's components.* To designate that the component is intended for only certain languages, click the ellipsis button (...) in this setting. The Languages dialog box opens, enabling you to select the appropriate languages for the component. For more information, see Marking Components as Language Dependent.  ***Tip*** • *To learn how to specify which language-dependent components are installed at run time, see Installing Components Based on Language.*
Releases view setting (Build tab)—Data Languages	Basic MSI, InstallScript MSI, Merge Module	If you want to include certain components and exclude others based on the language that is selected for each component, click the ellipsis button (...) in this setting and specify the appropriate languages. If the language specified for a component does not match one of the languages that is selected for this setting, InstallShield does not include the component in the release. By default, releases are language independent; that is, none of the project's components are excluded from the release.
Releases view setting (Build tab)—UI Languages	Advanced UI, Basic MSI, InstallScript MSI, Merge Module, Suite/Advanced UI	The UI Languages setting lets you specify which user interface languages you want to include in a release. To select the appropriate languages, click the ellipsis button (...) in this setting. Note that if a language is not selected in the Setup Languages setting in the General Information view of the project, it is not listed as one of the available languages for the UI Languages setting.

Table 9-1 • Language Settings in InstallShield (cont.)

Setting	Project Type	Description
Releases view setting (Build tab)—Language(s)	InstallScript, InstallScript Object	Use the Language(s) setting if you want to include certain components and exclude others based on the language that is selected for each component. This setting also lets you specify which user interface languages you want to include in a release. If the language specified for a component does not match one of the languages that is selected for this setting, InstallShield does not include the component in the release. In addition, if a UI language that is included in the project does not match one of the languages that is selected for this setting, InstallShield does not include the UI strings in the release. By default, releases are language independent; that is, none of the project's components or UI strings are excluded from the release. Note that if a language is not selected in the General Information view of a project, it is not listed as one of the available languages for the Language(s) setting.

For information about the default language in a project, see Setting the Default Project Language.

Selecting the Installation Languages



Edition • *The Premier edition of InstallShield includes support for creating multilingual installations.*

If you have the Premier edition of InstallShield, you can select the languages that you want to include in your installation. When you add a language to your project, InstallShield adds string entries for that language.

To add supported languages to your project, use the Setup Languages setting in the General Information view. The affect of this setting differs, depending on what project type you are using.

- In Basic MSI, InstallScript MSI, Merge Module, and Suite/Advanced UI projects, the Setup Languages setting in the General Information view lets you specify the languages that you want to be listed in the UI Languages setting in the Releases view. Therefore, if a language is not listed for this Setup Languages setting at the project level, you cannot include that particular UI language in your project's releases.
- In InstallScript and InstallScript Object projects, the Setup Languages setting in the General Information view lets you specify the languages that you want to be listed in the Languages setting in the Components and Releases views in your project. In general, if a language is not listed for this setting at the project level, you cannot designate that a particular component in your project is targeted for that language; in addition, you cannot designate that the components and UI strings for a particular language are included in your project's releases.

When you add a supported language to your project through this setting, InstallShield adds string entries for that language to your project. The string entries include the built-in user-interface string resources that are already translated.

The New Language Wizard in InstallShield lets you add unsupported languages to projects. An unsupported language is one in which none of the default run-time strings are translated. When you add an unsupported language to a project, that language is made available in the Setup Languages setting, and in other various language-related settings throughout InstallShield. In addition, InstallShield uses the strings from your project's default language as placeholders for the strings in that newly added unsupported language; you can use the String Editor view to provide translated strings for unsupported language.

You can build an installation for distribution in as many languages as you have support for, but InstallShield runs an installation in only one language. For more information, see How an Installation Determines Which Language to Use for the User Interface.

Marking Components as Language Dependent

All new components are language independent by default. Mark a component as specific to a language if you want the component and its data built into a package that targets that language.



Task ***To mark a component as language dependent:***

1. In the View List under Organization, click **Setup Design** (for installation projects only) or **Components**.
2. Select the component that you want to configure as language dependent. InstallShield displays the component's settings in the right pane.
3. Click the **Languages** setting. InstallShield displays the list of languages in the bottom-right pane.
4. Select the check box for each language to which this component's data apply. Clear the check box for each language that does not apply.



Tip *• To learn how to specify which language-dependent components are installed at run time, see Installing Components Based on Language.*

Installing Components Based on Language

Assuming that its feature is selected for installation, a language-dependent component that is present in the release is installed onto the target system. The component's language setting determines which components are built into the release, not necessarily which components are installed.

Specifying Which Language-Dependent Components Are Installed at Run Time for Basic MSI and InstallScript MSI Projects

To specify whether a component is installed based on the target system's language, use the Windows Installer property **SystemLanguageID** in the component's Condition setting.

For example, the following condition allows the component—which may or may not be marked as French in InstallShield—to be installed only onto French (France) systems:

```
SystemLanguageID = 1036
```

Other properties that can be used to determine language characteristics at run time are **UserLanguageID**, which is the numeric identifier of the end user's default language, and **ProductLanguage**, which is the identifier of the language in which the installation is running.

Specifying Which Language-Dependent Components Are Installed at Run Time for InstallScript Projects

During run time of InstallScript installations, you can control the languages that your installation supports by calling the **FeatureFilterLanguage** function.

In the **OnFilterComponents** event handler, the framework typically calls this function with the languages that match the target system so that only the appropriate components are installed. By calling **FeatureFilterLanguage**, you can override this default behavior to install or prevent the installation of components based on any language criteria that you specify.

For more information, see FeatureFilterLanguage.

Including Languages in the Release



***Edition** • The Premier edition of InstallShield includes support for creating multilingual installations.*

When you configure a release through the Release Wizard or the Releases view, you can specify the language of the installation's end-user interface and for filtering language-dependent application data.

You can use the same project to build a version of your release for any of the supported languages. Or you can select multiple languages in the Release Wizard or the Releases view to create a single installation that is capable of running in any one of the included languages and installing components specific to any number of languages.

Including User-Interface Resources for a Language

You decide which languages to make available in your installation in the Release Wizard or the Releases view.

While you can create a package that contains support for several languages, the installation itself is presented in only one language. For more information, see How an Installation Determines Which Language to Use for the User Interface.

Including Language-Dependent Components

Another consideration is which of your application's language-dependent data you want to include in your project. By marking a component as language dependent and then configuring the appropriate language-related setting on the Build tab in the Releases view as needed, you can automatically include or exclude (filter) that component from the release. You can also specify the language for the release in the Release Wizard.

If you do not select specific languages at the release level, InstallShield includes all components in the release. If you select a language, InstallShield builds all components that are specific to that language and all language-independent components into the release. For example, choosing to build an English-only release means that Japanese components—that is, components with Japanese selected for the Languages setting—are filtered out.

Tip *• You may have components specific to several languages depending on your selections for the language-related setting in the Releases view or the Release Wizard. To learn how to install them only on machines with a specific locale, see Installing Components Based on Language.*

Translating String Entries

Instead of hard-coding strings throughout your project, you can use string entries in areas of InstallShield that accept localizable text. Each string entry consists of a language-independent identifier and a corresponding language-specific value. At run time, the installation displays the appropriate translated string values.

To help streamline the process of localizing a project, all of the text strings that may be displayed at run time during the installation process are available in one consolidated view: the String Editor view. You can use this view to edit the strings for everything from button text to feature descriptions. You can also use this view to export each language's string entries to a file, translate the values that are listed in the file, and then import the translated file into your project.

For information on working with string entries, see Using String Entries in InstallShield.

For import and export instructions, see Exporting and Importing String Entries.

Exporting and Importing String Entries

To ease the task of translating all of the run-time strings in your project, InstallShield enables you to export the string entries for a language to a tab-delimited text (.txt) file. You can provide that .txt file to a translator who can update the file with translated text. Then you can import the .txt file back into your InstallShield project for a localized end-user interface.

Tip *• To automate the export and import process, you can use the InstallShield automation interface. To learn how, see Exporting and Importing String Entries Using the Automation Interface.*

Exporting a Language's String Entries from InstallShield to a Text File

InstallShield lets you export all of the string entries for a language, or only some of the string entries. You may want to export only the string entries that have been modified since a specific date. This is useful if you want to give the translator only the string entries that are new or have been modified since the last round of translations.

Task ***To export all of the string entries for a language:***

1. In the View List under **User Interface**, click **String Editor**.
2. Click the **Export Strings** button. The **Select File and Language to Export** dialog box opens.
3. Browse to the location where you want InstallShield to save the text file.
4. In the **File name** box, enter the name of the text file that you want InstallShield to create.
5. In the **Language** list, select the language whose strings you want to export to the text file.

6. Click **Save**.

InstallShield exports all of the selected language's string entries to a tab-delimited Unicode text file. You can give this text file to your translator.

Tip *• By default, the export process creates a Unicode text file. Although the* ***Select file and language to export*** *dialog box has a check box that lets you export the string entries to an ANSI file, an ANSI file is not created by default. The Unicode text file is the preferred type of file, since ANSI files requires that the build machine have the appropriate code pages for double-byte languages. To learn more, see Code Page Requirements for Language Support.*

Task ***To export only some of the string entries for a language:***

1. In the View List under **User Interface**, click **String Editor**.
2. Select the string entries that you want to export:
   - To select multiple consecutive rows, click the first row, and then press SHIFT while clicking the last row.
   - To select multiple nonconsecutive rows, click the first row, and then press CTRL while clicking each additional row.
3. Press CTRL+C to copy the string entries in the selected rows to the Clipboard.
4. Create a new text file in a text editor such as Notepad.
5. Press CTRL+V to paste the content in the Clipboard in the text file.
6. Save the text file. It is recommended that you save the file with Unicode encoding, since Unicode supports double-byte characters. To learn more, see Code Page Requirements for Language Support.

Tip *• The automation interface lets you automate the process of exporting only the string entries that have been modified since a specific date. To learn more, see Exporting and Importing String Entries Using the Automation Interface.*

Importing String Entries from a Text File Into an InstallShield Project

InstallShield lets you import translated string entries from a text file to an InstallShield project.

Task ***To import the tab-delimited text file that has been translated:***

1. In the View List under **User Interface**, click **String Editor**.
2. Click the **Import Strings** button. The **Select File and Language to Import** dialog box opens.
3. Select the text file that you want to import.
4. In the **Language** list, select the language for the text entries that you are importing.
5. Click **Open**.

The string entry importer checks for conflicts with string identifiers. If any conflicts exist, InstallShield prompts you to indicate whether you want to overwrite the existing entry in your project with the one in the text file.

Modifying Dialogs for Each Language

When you add support for a language to your project, InstallShield provides a version of each standard dialog translated into the newly added language. You can edit these dialogs, as well as custom or imported dialogs, for each supported language in the Dialog Editor.



Task ***To view these dialogs:***

1. In the View List under **User Interface**, click **Dialogs**.
2. In the **Dialogs** explorer, expand the **All Dialogs** item.
3. Double-click the name of a dialog to see an item for each supported language.
4. To modify a dialog's layout, select the language version.

The rule to remember is that all controls have the same properties, string entries, and behavior for each language. When you add a control to the English (United States) copy of a dialog, you are adding the same control to the German version. Setting the Sunken property of a bitmap to True makes that property True for each language-specific version.

Resizing Elements

The exceptions to the above rule are the Height and Width properties of a control. These properties are specific to each language's version. Because string lengths can vary widely from one language to another, when you resize a control, you are not affecting the control's size for any other language.

For example, you may need to enlarge a push button to accommodate a longer string after it is translated into German. The push button remains the same size for every version of the dialog since it was created. When you resize the control in the German layout, it is resized only for that language.

Modifying Strings

The strings in each dialog come from that language's string entries. When you select a string for a control that accepts localizable text, although the string identifier is the same for each language-specific version, the string value that is displayed comes from the current language's string entries. If you edit the value in the control's property sheet, you are actually editing that string identifier's value for that language.

Modifying File Resources

Several controls, such as bitmaps and check boxes, accept a file that will be streamed into the setup package. Because the file resources can be different depending on the language, when you edit the File Name value, you will be supplying a file only for that particular language.

The file name can differ for each language because every File Name property uses a string entry. Thus, the file name is originally the same for every language. When you edit strings in the String Editor view or edit the dialog layout for a specific language, you can enter a new file name only for that language.

Right-to-Left-Language Dialogs

Basic MSI and Merge Module projects include support for languages that are written and read from right to left. For more information, see Dialog Support for Right-to-Left Languages.

How an Installation Determines Which Language to Use for the User Interface

Although you can localize an installation in as many languages as your project supports, an installation can be run in only one language. When **Setup.exe** initializes, it determines which language should be used for the installation.

Determining the Language at Setup.exe Initialization

If your installation has support for more than one language (that is, if you selected more than one language in the Setup Languages panel of the Release Wizard or on the Build tab for the release in the Releases view), and if one of those languages matches one of the following items on a target system, **Setup.exe** launches the installation in the matching language. Note that the installation evaluates the following items in the order that they are listed and uses the first language that matches:

1. The language that the end user specifies through the /L command-line parameter
2. If an earlier version of the product was installed on the target system and is still present, the language that was used to install the earlier version
3. The user default language of the target system
4. The system default language of the target system
5. The system default UI language of the target system
6. The default language for the installation, which you can set in the Setup Languages panel of the Release Wizard, or on the Build tab for the release in the Releases view

If your installation has support for only one language, **Setup.exe** launches the installation in that language.

Displaying the Language Selection Dialog at Run Time

InstallShield lets you specify whether you want **Setup.exe** to display the Languages dialog that allows end users to choose the language in which the installation should run. The dialog presents a list of available languages, which are the same languages that you select in the Setup Languages panel of the Release Wizard, or on the Build tab for the release in the Releases view. The text on the Languages dialog is displayed in the language that **Setup.exe** selects at initialization.

When the end user makes a selection on the Languages dialog, a Basic MSI or InstallScript MSI installation applies a transform that contains all of the user interface resources for that language and then launches the installation in the selected language.

In an InstallScript installation, the user interface resources are stored in the _isres*.dll files. The strings are built into string table files that are included with the installation's support files, along with the _isres*.dll files.

Because the Languages dialog is displayed by **Setup.exe**, you need to create a setup launcher if you want the Languages dialog to be included in the user interface of your installation. For more information, see Creating a Setup Launcher.



Note *• The first time that the end user selects a language in the Languages dialog, the installation runs in the selected language. After an installation has been run in a particular language, it cannot be run in any other language on the same machine. This prevents the installation from running repair mode in a language that is different from the one that was used for the first-time installation. It also prevents the installation from installing features for a language that is different from the one that was used for the first-time installation.*

Customizing Language Support



Edition *• The New Language Wizard is available in the Premier edition of InstallShield.*



Project *• The New Language Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

If you need your installation to run in languages that are not supported by InstallShield Premier Edition, or you want to create your own translations for some of the supported languages, you can add support for those languages with the New Language Wizard. This wizard lets you to select the languages that you would like to support and the projects to which you would like to add these languages. It then adds the languages you choose to the list of available languages for your installation.

To learn more about this wizard, see New Language Wizard.

Language Identifiers

Throughout the InstallShield interface, you must refer to specific languages by their language identifiers, or LCIDs. The identifier is an integer value that identifies a specific language. The language ID is more commonly given as a hexadecimal value, but you must specify the decimal version for Windows Installer.

The following table shows the LCIDs for all of the supported languages. To learn more about language support, see Run-Time Language Support in InstallShield.



Project *• Note that Basic MSI and InstallScript MSI installations use different IDs than InstallScript installations.*



Edition • *Also note that support for Arabic (Saudi Arabia) and Hebrew is available only in Basic MSI, Merge Module, and Suite/Advanced UI projects in the Premier edition.*

Table 9-2 • Language Identifiers

Language Name	Identifier (InstallScript Projects)	Identifier (Windows Installer–Based Projects and Suite/Advanced UI Projects)
Arabic (Saudi Arabia)	(This language is not supported.)	1025
Basque	0x042d	1069
Bulgarian	0x0402	1026
Catalan	0x0403	1027
Chinese (Simplified)	0x0804	2052
Chinese (Traditional)	0x0404	1028
Croatian	0x041a	1050
Czech	0x0405	1029
Danish	0x0406	1030
Dutch	0x0413	1043
English (United States)	0x0409	1033
Finnish	0x040b	1035
French (Canada)	0x0c0c	3084
French (France)	0x040c	1036
German	0x0407	1031
Greek	0x0408	1032
Hebrew	(This language is not supported.)	1037
Hungarian	0x040e	1038
Indonesian	0x0421	1057

Table 9-2 • Language Identifiers (cont.)

Language Name	Identifier (InstallScript Projects)	Identifier (Windows Installer–Based Projects and Suite/Advanced UI Projects)
Italian	0x0410	1040
Japanese	0x0411	1041
Korean	0x0412	1042
Norwegian	0x0414	1044
Polish	0x0415	1045
Portuguese (Brazil)	0x0416	1046
Portuguese (Portugal)	0x0816	2070
Romanian	0x0418	1048
Russian	0x0419	1049
Serbian (Cyrillic)	0x0c1a	3098
Slovak	0x041b	1051
Slovenian	0x0424	1060
Spanish	0x040a	1034
Swedish	0x041d	1053
Thai	0x041e	1054
Turkish	0x041f	1055

Installing Multiple Instances of Products

Project *• This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

Caution *• Creating an installation that lets end users install multiple instances of a product on the same machine and in the same context requires sophisticated authoring and serious commitment on the part of the installation developer. This functionality is recommended for only advanced installation developers.*

Windows Installer allows only one instance of a product code to be installed in the machine context and only one instance to be installed in each user context. Windows Installer 3.x and later includes support for a product code–changing transform. This type of transform—called an *instance transform*—enables the same .msi package to be used to install multiple instances of the same product in the same context because it changes the product code for each instance.

Using the multiple-instance support in InstallShield reduces some of the effort needed to support multiple instances of a product. With InstallShield, you can configure one base installation or patch for a product and then configure multiple instances that correspond with each additional instance that you want to support. At build time, InstallShield creates an instance transform for each instance and streams the instance transforms into the .msi package. At run time, the installation typically displays an instance selection dialog that lets end users specify whether they want to install a new instance or maintain an existing one.

Run-Time Requirements for Multiple-Instance Support

Project *• This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

The following table lists the operating system and Windows Installer requirements for multiple-instance support.

Table 9-1 • Run-Time Requirements for Multiple-Instance Support

Type of Requirement	Requirement	Implication
Target operating system	The target system must have one of the following operating systems, or a later version: • Windows Server 2003 • Windows Vista • Windows XP SP1	Consider adding a launch condition so that end users can install the product only on platforms that support installing multiple instances of products.  ***Tip*** • *You can use the Installation Requirements page of the Project Assistant to quickly add launch conditions to your project. For more information, see Specifying Operating System Requirements in the Project Assistant.*
Windows Installer version	Windows Installer 3.0 or later is required.	If target systems may not have Windows Installer 3.0 or later, consider adding the Windows Installer redistributable to your project. To learn more, see Adding Windows Installer Redistributables to Projects.

Configuring Multiple Instances in InstallShield



Project • *This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

The typical process for adding multiple-instance support to a Basic MSI project is as follows:

1. For a product configuration in the Releases view, add a new instance for each instance that you want to support.
2. Set the properties for each instance.
3. Ensure that each instance will have its own isolated files and nonfile data on the target system if appropriate. For more information, see Special Considerations for Multiple-Instance Support.
4. Build your release.
5. Thoroughly test your release.

Adding an Instance to a Product Configuration

Project *• This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

When you are configuring multiple instances for your product in InstallShield, you need to add and define one instance for each instance to be installed, in addition to the base instance that is permitted by your base installation package. In InstallShield, instances are defined at the product configuration level. Each instance that you create corresponds with a different instance transform that InstallShield creates at build time.

Note *• Do not add an instance for the base instance that is permitted by your installation package. Your installation does not need an instance transform for this instance.*

Task ***To add an instance to a product configuration:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the product configuration that should contain the new instance.
3. Click the **Multiple Instances** tab.
4. Right-click the **Instances** explorer and then click **New**.

InstallShield adds a new instance with a number as the name of the instance. In addition, InstallShield adds the **ProductCode** property, with a new GUID value, to the pane on the right.

Once you have added an instance, you need to set its properties. For more information, see Setting Properties for an Instance.

Tip *• When InstallShield adds the new instance to the Instances explorer, you can enter a new name for the instance, or right-click the instance later and then click Rename. For information on naming an instance, see Naming an Instance.*

Naming an Instance

Project *• This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

When you add the first instance to any product configuration in your project, InstallShield adds the Windows Installer property **InstanceId** to your project, and sets its value to 0. The 0 value is for the base installation package. Each instance that you define in your project must have a different integer for its **InstanceId** value.

InstallShield uses the name of an instance on the Multiple Instances tab in the Releases view as the **InstanceId** property value that corresponds with that instance. Therefore, when you are naming an instance, make sure that you conform to the following guidelines:

- Each instance in a product configuration must have a different name.
- Each instance name must be an integer: it cannot contain any letters or other characters.

Task ***To assign a new InstanceId value to an instance:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the product configuration that contains the instance that you want to modify.
3. Click the **Multiple Instances** tab.
4. In the **Instances** explorer, right-click the instance whose **InstanceId** property value you want to change, and then click **Rename**.
5. Enter a new value.

Updating the Product Code of One or More Instances

Project • *This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

Each instance must have a unique product code. Therefore, when you add an instance to a product configuration in the Releases view, InstallShield automatically adds the **ProductCode** property, with a new GUID value, to the pane on the right for that instance.

InstallShield enables you to easily change the GUID value of the **ProductCode** property of one or more instances that are defined for a product configuration in your project.

Task ***To update the product code of a specific instance:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the product configuration that contains the instance.
3. Click the **Multiple Instances** tab.
4. In the **Instances** explorer, click the instance whose product code you want to modify.
5. In the grid in the right pane, right-click the GUID for the ProductCode setting that you want to update, and then click **Generate a New GUID**.

InstallShield changes the GUID.

Task ***To update the produce code of each instance that is defined on the Multiple Instances tab for a product configuration:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the product configuration that contains the instance.
3. Click the **Multiple Instances** tab.
4. Right-click the **Instances** explorer and then click Change ProductCode of Each Instance.

InstallShield changes the GUID of each instance that is defined for the product configuration.

Setting Properties for an Instance

Project *• This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

Each instance must have a unique product code. Therefore, when you add an instance to a product configuration in the Releases view, InstallShield automatically adds the **ProductCode** property, with a new GUID value, to the pane on the right for that instance.

You should also set the **ProductName** property to a different name for each instance. This helps to make instances easily distinguishable from each other in Add or Remove Programs.

You can set additional properties for an instance. For example, you may want to define instance-specific properties that you use in conditions for components. As another example, you may want to set a property for each instance to define an instance-specific location for certain files or registry entries.

If you want to create major upgrades that can be applied to each individual instance, set the **UpgradeCode** property to a different GUID for each instance. Note that when you add a major upgrade item to your project in the Upgrades view, ensure that you select the **Products sharing my Upgrade Code** option on the Common tab for the major upgrade item.

Tip *• InstallShield automatically sets the value of the* ***InstanceId*** *property with a different identifier for each instance in a product configuration. To learn more, see Naming an Instance.*

Task ***To set a property for an instance:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the product configuration that contains the instance.
3. Click the **Multiple Instances** tab.
4. In the **Instances** explorer, click the instance.

5. In the grid in the right pane, click the field in the **Property** column of the last row, and then enter the name of the property that you want to set.
6. In the **Value** column, enter the property value that you want to be associated with the current instance.

To set the value of a property for the base instance—the instance that is installed by the base installation package—enter the property and corresponding value in the Property Manager view. To learn more, see Creating Properties in Windows Installer–Based Projects.

Deleting an Instance from a Product Configuration



Project • *This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

If you want to reduce the number of instances of your product that are allowed in the machine context and in each user context, you can delete an instance from a product configuration in your project.



Task ***To delete an instance from a product configuration:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the product configuration that contains the instance that you want to remove.
3. Click the **Multiple Instances** tab.
4. In the **Instances** explorer, right-click the instance that you want to remove, and then click **Delete**.

InstallShield deletes the instance, as well as all of its associated properties, from your project.

Special Considerations for Multiple-Instance Support



Project • *This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.



Caution • *Creating an installation that lets end users install multiple instances of a product on the same machine and in the same context requires sophisticated authoring and serious commitment on the part of the installation developer. This functionality is recommended for only advanced installation developers.*

Supporting the installation of multiple instances of a product requires careful advance planning to ensure that the multiple instances can exist side by side within the same context. Consider the following points when you are planning your installation.

- To ensure that uninstalling or upgrading some instances of your product does not affect other instances, you may need to isolate some or all of the files and the nonfile data that are installed on the target system for each instance.
- In some cases, you may not want to isolate some of the files. For example, if your installation includes one or more COM servers, you may want to configure each component that contains a COM server to install to a location that does not change for each instance. For these components, you can select Yes for the Shared setting so that Windows Installer uses DLL reference counting.
- You may need to create a separate set of components for each instance that your installation supports. In this scenario, you could add a condition in the Components view or the Setup Design view for each component so that each one is installed for the appropriate instance. For example, since the default **InstanceId** value for the base instance that is installed by the installation is 0, you could use the following condition for a component that contains data for the base instance:

  InstanceId=0

  The default **InstanceId** value for the next instance that is installed is 1. Therefore, you could use the following condition for a component that contains data for this next instance:

  InstanceId=1

For more information, see Authoring Multiple Instances with Instance Transforms in the Windows Installer Help Library.

Configuring and Building a Release that Includes Multiple-Instance Support



Project • *This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

Configuring the Release

To include multiple-instance support in a release, you must configure your release to include a **Setup.exe** setup launcher, since it displays the instance selection dialog when appropriate. To learn how to include the **Setup.exe** setup launcher, see Creating a Setup Launcher.

Building the Release

Building a release that includes multiple-instance support is slightly different than building a release that does not include this support. For multiple-instance support, InstallShield builds some additional files: instance transforms and instance packages.

Instance Transforms

At build time, InstallShield generates instance transforms for each instance. InstallShield streams the instance transforms into the .msi file that it creates in the Disk1 folder. Depending on how you have configured the release, the .msi file may be compressed into the **Setup.exe** file, or it may be left uncompressed.

Instance Packages

At build time, InstallShield generates an .msi file for each instance. The name of each file is **InstanceIdN.msi**, where **N** represents the value of the **InstanceId** property for that instance. The .msi files are stored in a folder called *Instances* under the release location.

Configuring and Building a Major Upgrade that Includes Multiple-Instance Support



Project • *This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

Configuring the Release

To include multiple-instance support in a release, you must configure your release to include a **Setup.exe** setup launcher, since it displays the instance selection dialog when appropriate. To learn how to include the **Setup.exe** setup launcher, see Creating a Setup Launcher.

Configuring the Upgrade

The following requirements must be met to have major upgrade support for multiple instances:

- Each instance in your project must have an UpgradeCode property with its own value. To learn how to define an UpgradeCode property for an instance in your project, see Setting Properties for an Instance.
- Each instance of a newer version must have a ProductCode property value that is different than the same instance of a related version.
- Each instance of a newer version must have a different product version (a larger number) than the same instance of a related version.
- The **Products sharing my Upgrade Code** option must be selected for the major upgrade in the Upgrades view.

Building the Release

Building a release that includes multiple-instance support is slightly different than building a release that does not include this support. For multiple-instance support, InstallShield builds some additional files: instance transforms and instance packages.

Instance Transforms

At build time, InstallShield generates instance transforms for each instance. InstallShield streams the instance transforms into the .msi file that it creates in the Disk1 folder. Depending on how you have configured the release, the .msi file may be compressed into the **Setup.exe** file, or it may be left uncompressed.

Instance Packages

At build time, InstallShield generates an .msi file for each instance. The name of each file is **InstanceIdN.msi**, where **N** represents the value of the **InstanceId** property for that instance. The .msi files are stored in a folder called *Instances* under the release location.

Creating Patches for Multiple Instances of a Product



Project • *This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

To create a patch that updates multiple instances of a product, begin by creating a minor upgrade or a small update. Then configure a patch in the Patch Design view. For the previous setup, select the **Setup.exe** file or the .msi file of the earlier installation that installs the version of the product that you want to upgrade. For the latest setup, select the **Setup.exe** file or the .msi file of upgrade release.



Tip • *For detailed information about creating upgrades and patches, see Updating Applications.*

At patch build time, InstallShield creates an **Update.exe** file or an .msp file, depending on how you have configured the patch in the Patch Design view.

For an overview on the run-time behavior of a patch that includes multiple-instance support, see Run-Time Behavior for Installing Multiple Instances of a Product.

Run-Time Behavior for Installing Multiple Instances of a Product



Project • *This information applies to Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

Running a First-Time Installation that Includes a Setup.exe File

The first time that an end user runs the **Setup.exe** file for your installation that supports multiple instances, Windows Installer installs the base instance that corresponds with your base installation package without any of the instance transforms. By default, the standard dialogs are displayed, regardless of whether your installation supports multiple instances.

If your product is successfully installed and the end user runs your installation again, the setup launcher displays a dialog that lets them select which instance they want to install. This instance selection dialog is displayed after the language selection dialog, if your installation includes one, but before the InstallWelcome dialog. The dialog contains two radio buttons:

- **Install a new instance**—This option lets end users install a new instance of your product. When an end user selects this option, the new instance of the product is installed.
- **Maintain or upgrade an existing instance**—This option lets end users maintain or upgrade an instance that they select from a list of already installed instances. When an end user selects this option, selects the instance that they want to change, and then clicks Next, the installation displays the following dialogs for the selected instance: the PreparingToInstall dialog, the MaintenanceWelcome dialog, and then the MaintenanceType dialog.

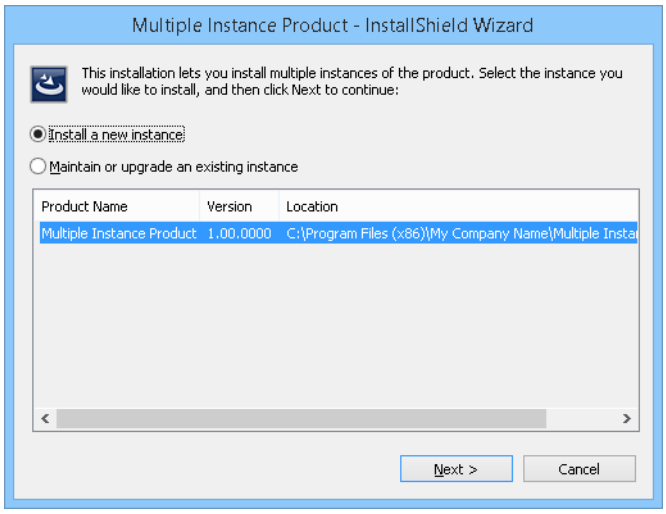


Figure 9-1: Instance Selection Dialog Displayed the Second Time the Product Is Being Installed

If an end user installs the last supported instance and then runs the installation again, the same dialog is displayed, but the **Install a new instance** option is disabled. The **Maintain or upgrade an existing instance** option is selected, and the end user can select the instance that they want to maintain or upgrade.

Tip • *To suppress the instance selection dialog, you can use the /instance command-line parameter.*

Running an Upgrade Packaged as a Full Installation that Includes a Setup.exe File

When a multiple-instance installation is an upgrade that is packaged as a full installation and that has a **Setup.exe** file, it behaves as a first-time installation if an end user is installing a new instance by launching the **Setup.exe** file. If an end user selects the **Maintain or upgrade an existing instance** option for an already installed instance, the installation lets the end user upgrade or maintain that instance, just as it would for a first-time installation.

Tip • *To suppress the instance selection dialog, you can use the /instance command-line parameter.*

Launching a First-Time Installation or an Upgrade that Is Packaged as a Full Installation Without a Setup.exe File from the Command Line

When you do not include a **Setup.exe** setup launcher with your multiple-instance installation, an end user can run your .msi file to install the base instance that corresponds with your base installation package without any of the instance transforms.

In order to install any instance that corresponds with one of the instance transforms in your package, the end user needs to include the appropriate command-line parameters for **MsiExec.exe**.

For example, the following command line installs a new instance, since the **MSINEWINSTANCE** property is set to 1. Windows Installer applies the **InstanceId1.mst** instance transform during installation of the new instance. The colon before the name of the instance transform is required because InstallShield embeds the instance transforms within the .msi file.

```
msiexec /i MyPackage.msi MSINEWINSTANCE=1 TRANSFORMS=:InstanceId1.mst
```

Note that the convention that InstallShield uses for naming instance transform files is the property name *InstanceId*, followed by the instance's value of the **InstanceId** property, followed by *.mst*. Thus, for the instance whose **InstanceId** property is set to 5, the name of the transform is **InstanceId5.mst**.

The **MSINEWINSTANCE** property should be set to 1 only if it is the first time that the end user is installing the specified instance; otherwise, an error is displayed.

To perform maintenance for an instance or to upgrade an instance, use the `/n` parameter to pass the instance's product code. For example, the following command line displays the MaintenanceType dialog, which lets the end user specify whether they want to upgrade, maintain, or remove the instance of the product that has the specified product code:

```
msiexec /i MyPackage.msi /n {00000001-0002-0000-0000-624474736554}
```

To uninstall an existing instance, use the following format at the command line:

```
msiexec /i MyPackage.msi /n {00000001-0002-0000-0000-624474736554} /x
```

Running an Upgrade that Is Packaged as a Patch with an Update.exe File

If you package your upgrade as a patch and you specify that you want to include an **Update.exe** file, the installation displays the patch version of the instance selection dialog. The dialog contains two radio buttons:

- **Patch all of the existing instances**—This option lets end users apply a patch to all installed instances of your product. When an end user selects this option, Windows Installer applies the patch to each instance separately, until all instances are updated. The instances are updated in order from the lowest **InstanceId** property value to the highest.
- **Patch an existing instance**—This option lets end users apply a patch to an instance that they select from a list of already installed instances. When an end user selects this option, selects the instance that they want to change, and then clicks Next, the installation displays the PreparingToInstall dialog and then the PatchWelcome dialog.



Tip *• To suppress the patch version of the instance selection dialog, you can use the /instance command-line parameter.*

Running an Upgrade Packaged as a Patch Without an Update.exe File

If you use an .msp file without an **Update.exe** file for your patch, and the end user runs your .msp file without passing any command-line parameters, Windows Installer behaves as if the end user selected the **Patch all of the existing instances** option in the patch version of the instance selection dialog: Windows Installer applies the patch to each instance separately, until all instances are updated. The instances are updated in order from the lowest **InstanceId** property value to the highest.

To apply an .msp patch to a specific instance, pass the /p option, as well as the /n option. The /n option must specify the product code of the installed instance to which the patch should be applied. For example:

```
msiexec /p mypatch.msp /n {00000001-0002-0000-0000-624474736554}
```

Detecting Conditions on the Target System

When you are creating an installation, you may need to make sure that certain conditions exist on the target system. For example, if your product requires a particular operating system, your installation may need to check the target system to ensure that this requirement is met. If the requirement is not met, your installation might display an error to inform the end user about the requirement.

InstallShield includes support for detecting various conditions on a target system. For more information, see this section of the documentation.

Building Conditional Statements



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

A conditional statement is an expression that the Windows Installer can evaluate as True or False. Typically, conditional statements are used to perform some action or enable a component's installation based on the value or existence of a property.

You can use operators similar to those in Visual Basic for comparative and logical operations. For more information, see Conditional Statement Syntax.



Tip • *In the Components, Dialogs, Custom Actions and Sequences, and Custom Actions views of InstallShield, you can access the Condition Builder dialog box when you click the ellipsis button in the condition property to help you build conditional statements.*



Important • *When you click the OK button on this dialog box, InstallShield performs basic condition validation; however, you should still double-check that your condition statements evaluate to the expected outcome. For more information and example conditions, see Conditional Statement Syntax.*

Examples

The following table provides examples of some conditional statements and explanations of how they can be used. Consult the Windows Installer Help Library for more information on Windows Installer properties and conditional expression operators.

Table 9-1 • Examples of Conditional Statements

Purpose	Syntax
To install a component only on machines that have the client version of Windows 7 and later, specify this expression for its Condition setting.	VersionNT>=601 And MsiNTProductType=1
To install a component only on Windows Server 2008 R2 and later domain controllers and servers, specify this expression for its Condition setting.	VersionNT>=601 And MsiNTProductType>1
To install a component only on Windows XP, specify the expression for its Condition setting.	VersionNT=501  ***Note*** • *You can also use the ServicePackLevel and WindowsBuild properties to distinguish between different operating systems. For a list of values, see Operating System Property Values in the Windows Installer Help Library.*
To install a component only on machines running Windows 7 or later, specify this expression for its Condition setting.	VersionNT>=601
To execute a custom action only if the product has not been installed, check the value of the Installed property with this condition.	Not Installed
To install a component only if the target system is using U.S. English as the default language, compare the value of the SystemLanguageID property to the language ID for U.S. English by using this component condition.	SystemLanguageID=1033  ***Note*** • *Related properties include* ***UserLanguageID*** *(the default language for the current user) and* ***ProductLanguage*** *(the language being used for the setup program's user interface).*
To allow your product to be installed only if the screen resolution is a minimum of 1024 X 768, add this condition to the Install Condition setting in the General Information view.	ScreenX >= 1024 AND ScreenY >= 768

Table 9-1 • Examples of Conditional Statements (cont.)

Purpose	Syntax
To allow your product to be installed only if the target system has at least 2 GB of RAM, add this condition to the Install Condition setting in the General Information view.	PhysicalMemory >= 2048

Conditional Statement Syntax



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

Conditional statements have a strictly governed syntax that uses properties, Windows Installer table keys, literals, and evaluative operators.

Functionality

A conditional statement performs some action in an installation—such as launching a custom action or installing a component—based on whether the expression evaluates to True, which has the numeric value of 1. False returns have the numeric value of 0. The simplest of expressions, specifying a property name, is True if the property is defined. For example, the simple conditional statement `Version9X` evaluates to True when the installation is running under Windows 95 or Windows 98.

You can build more complex conditional statements using the operators and values described in the tables below.

Values

The following table describes how to use values in your conditional statements. All values except for environment variables are case sensitive.

Table 9-2 • Descriptions of Values

Value	Description
Windows Installer property	The name of the property. Nonexistent properties are evaluated as empty strings.
Integer	Any integer value from -32,767 to +32,767. You cannot use a floating point value.

Table 9-2 • Descriptions of Values (cont.)

Value	Description
String literal	Enclose the text in quotation marks. For example: **"InstallShield"**
Environment variable	Precede the variable name with a percent symbol. For example: **%TEMP**

Operators

The tables below list all of the conditional statement operators by type. Note the following:

- The operators follow the same precedence as those in Visual Basic.
- Use parentheses to override precedence.
- Operators are not case sensitive.
- Unlike in Visual Basic, there are no arithmetic operators.
- Precede the operator with a tilde (~) if you do not want the comparison to be case sensitive.
- The result is always False when you compare a string with an integer, unless you use the <> operator.

Table 9-3 • Descriptions of Logical Operators

Logical Operator	Description
Not	Performs logical negation. The result is True if the value is False, and vice versa.
And	The result is True if both values are True.
Or	The result is True if either value is True.
Xor	The result is True if only one of the values is True.
Eqv	The result is True if both values are True or both values are False.
Imp	Performs a logical implication. The result is True if the first term is False or the second is True.

Table 9-4 • Descriptions of Comparative Operators

Comparative Operator	Description
=	The result is True if the first value is equal to the second.
<>	The result is True if the first value is not equal to the second.

Table 9-4 • Descriptions of Comparative Operators (cont.)

Comparative Operator	Description
>	The result is True if the first value is greater than the second.
>=	The result is True if the first value is greater than or equal to the second.
<	The result is True if the first value is less than the second.
<=	The result is True if the first value is less than or equal to the second.

Table 9-5 • Descriptions of Substring Operators

Substring Operator	Description
><	The result is True if the first string contains the second.
<<	The result is True if the first string starts with the second.
>>	The result is True if the first string ends with the second.

Table 9-6 • Descriptions of Bitwise Operators

Bitwise Operator	Description
><	The result is True if both integers have any bits in common.
<<	The result is True if the high 16 bits of the first integer are equal to the second integer.
>>	The result is True if the low 16 bits of the first integer are equal to the second integer.

Detecting Administrator and Elevated Privileges

Basic MSI Projects

On systems with Windows XP and earlier or Windows Server 2003 and earlier, two properties for detecting user privileges are **AdminUser** and **Privileged**. The **AdminUser** property is set if the end user has Administrator privileges. The **Privileged** property is set if the installation is running with elevated privileges (that is, if the end user has administrative privileges, if the installation has been assigned by a system administrator, or if both the user and machine AlwaysInstallElevated policies are set to true). In most cases, the **Privileged** property is more appropriate.

On Windows Vista and later and Windows Server 2008 and later, the **AdminUser** property is by default assigned the same value as the **Privileged** property. To restore the distinction between **AdminUser** and **Privileged** on these systems, you can set the **MSIUSEREALADMINDETECTION** property to 1 in the Property Manager view.

Note that for Windows Vista and later and Windows Server 2008 and later, **AdminUser** and **Privileged** are always set during the User Interface sequence; therefore, they cannot detect whether an installation is actually running with elevated privileges during the User Interface sequence. However, custom actions running as deferred in system context have the correct value for **Privileged** (and for **AdminUser**, if **MSIUSEREALADMINDETECTION** is also set).

Because only actions running as deferred in system context should modify the system, distinguishing privileged from non-privileged installations is significant only for that type of action. One consequence of this behavior is that **AdminUser** and **Privileged** should not be used in a project's install conditions for targeting Windows Vista and later or Windows Server 2008 and later.

InstallScript and InstallScript MSI Projects

The following InstallScript expression returns TRUE if the end user has Administrator privileges, except for some cases on Windows Vista or later systems and Windows Server 2008 or later systems:

```
Is(USER_ADMINISTRATOR, "");
```

On Windows Vista and later systems and Windows Server 2008 and later systems, **Is** returns TRUE if the SE_GROUP_USE_FOR_DENY_ONLY security identifier (SID) attribute is not set for the group. That is, if the current user is in the Administrators group but that user is running the installation with a standard access token on Windows Vista, **Is** returns FALSE.

Detecting First-Time Installations, Maintenance Mode, and Uninstallation

Your installation can determine whether the installation is being run for the first time on a target system.

Windows Installer–Based Projects

In the Windows Installer sequences, the following conditions detect certain types of installation:

- First-time installation: Not Installed
- Maintenance: Installed
- Uninstallation: REMOVE="ALL" (after the InstallValidate action)

You can use these conditions in any of the Condition settings of a Basic MSI or InstallScript MSI project.

InstallScript-Based Projects

In InstallScript, the MAINTENANCE variable is FALSE for a first-time installation, and TRUE for maintenance mode or uninstallation. Therefore, you can use an if-statement like the following for any code you want to run only for a first-time installation.

```
if (!MAINTENANCE) then
    // code to run for first-time installation
endif;
```

You can use that sort of code in event-driven InstallScript code of InstallScript and InstallScript MSI projects.

Triggering Behavior Only During a First-Time Installation

There may be times when you want certain behavior to occur only during a first-time installation, but not during uninstallation.

Basic MSI Projects

At run time for Basic MSI projects, the same sequences are used for first-time installations and maintenance installations (including uninstallation). There is no separate uninstallation sequence. Therefore, any custom actions that you schedule in the Installation sequences will, by default, run for both installation and uninstallation.

To specify that an action should run only for a first-time installation, you can use the Not Installed condition. The Not Installed condition is appropriate, for example, for a custom action that launches a Readme file for the application being installed.

InstallScript and InstallScript MSI Projects

In InstallScript and InstallScript MSI installations, you can use the MAINTENANCE system variable to determine whether the installation is running in maintenance mode or if it is a first-time installation. The following code sample shows how to use it in your script:

```
if (!MAINTENANCE) then
    // it is a first-time install
endif;

And for reinstall/modify/repair, it's:

if (MAINTENANCE) then
    // ...not first-time
endif;
```

Detecting If the End User Has Selected a Specific Feature

Your installation can determine whether the end user has selected to install a specific feature. The method depends on which project type you are using.

Basic MSI Projects

Windows Installer defines conditions of the form &FeatureName=*n* to detect a feature's action state. For example, to determine if a feature called ProgramFiles is selected to be installed locally, and it was not already installed, use the condition &ProgramFiles=3.

InstallScript and InstallScript MSI Projects

The InstallScript function **FeatureIsItemSelected** returns TRUE if a feature is currently selected in one the feature-selection dialogs (such as **SdFeatureTree**).

Detecting If the End User Is Running a Particular Operating System

Your installation can determine which operating system is on a target system.

Using Windows Installer Properties to Detect the Operating System

The Windows Installer properties **VersionNT**, **Version9X**, **ServicePackLevel**, and **WindowsBuild** describe the target operating system.



***Project** • You can use these Windows Installer properties in conditions for items such as features, components, and custom actions in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*



***Note** • Property names are case-sensitive, so **Version9X** and **Version9x** are considered different properties.*

Using an InstallScript Structure Variable to Detect the Operating System

The InstallScript structure SYSINFO is automatically initialized to describe the operating system on the target system. You can use this structure in your InstallScript code to trigger specific behavior on certain platforms. For specific values to check, see SYSINFO.



***Project** • You can use the SYSINFO structure in event-driven InstallScript code in the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

You can also use this structure in InstallScript custom actions in the following project types:

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

Detecting Whether the Installation Is Being Run on a Virtual Machine

InstallShield lets you determine whether an installation is running on any of the following types of virtual machines:

- Microsoft Hyper-V
- A VMware product such as VMware Player, VMware Workstation, or VMware Server
- Microsoft Virtual PC

To check for virtual machines, you can either use Windows Installer properties, or you can use an InstallScript structure or function.



***Note** • Emulators such as Bochs and QEMU are not detected.*

Using Windows Installer Properties to Detect a Virtual Machine

Project *• You can use Windows Installer properties in conditions for items such as features, components, and custom actions in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

To use Windows Installer properties to detect the presence of a virtual machine and determine the type of virtual machine, you first need to create a custom action that calls the ISDetectVM function in the **SetAllUsers.dll** file.

Task ***To add a custom action that checks for a virtual machine:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI and InstallScript MSI projects) or **Custom Actions** (in DIM and Merge Module projects).
2. In the center pane, right-click the **Custom Actions** explorer, point to **New MSI DLL**, and click **Stored in Binary table**. InstallShield adds a new custom action.
3. Change the name of the action to something like **ISDetectVM**.
4. Configure the custom action's settings in the right pane:
   - In the **DLL Filename** setting, select the **SetAllUsers.dll** file. The typical path is:

     <ISRedistPlatformDependentFolder>\SetAllUsers.dll
   - In the **Function Name** setting, enter the following name:

     ISDetectVM
   - In the **In-Script Execution** setting, select **Immediate Execution**.
   - Schedule the custom action as needed by selecting the appropriate option in one or more sequence settings.

The custom action sets the following Windows Installer properties at run time:

Table 9-7 • Windows Installer Properties for Detecting Virtual Machines

Property Name	Description
IS_VM_DETECTED	If the value of this property is 1, the installation is running on one of the following virtual machine environments: • Microsoft Hyper-V • A VMware product such as VMware Player, VMware Workstation, or VMware Server • Microsoft Virtual PC If the value is not set, no virtual machine has been detected.

Table 9-7 • Windows Installer Properties for Detecting Virtual Machines

Property Name	Description
IS_VM_TYPE	This property indicates the type of virtual machine that is running the installation. Available values are: • **Undefined**—No virtual machine has been detected. • **1**—The installation is running on a VMware product such as VMware Player, VMware Workstation, or VMware Server. • **2**—The installation is running on a Microsoft Hyper-V machine. • **3**—The installation is running on a Microsoft Virtual PC machine. • **4**—The type of virtual machine is not known.

You can use the aforementioned properties in conditional statements. For example, if your product cannot be used on virtual machines, enter the following conditional statement in the Install Condition setting of the General Information view:

Not IS_VM_DETECTED

If you have a component that should be installed only on VMware images, enter the following conditional statement in the Condition setting of the Components view:

IS_VM_TYPE=1

Using an InstallScript Structure and Variable to Detect a Virtual Machine



Project • *You can use an InstallScript structure and variable in event-driven InstallScript code in the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

You can also use the structure and variable in InstallScript custom actions that are in the following project types:

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

You can use the SYSINFO.bIsVirtualMachine structure in your InstallScript code to determine whether the installation is being run on a virtual machine. In addition, you can use the VIRTUAL_MACHINE_TYPE constant with the **GetSystemInfo** function to determine which type of virtual machine is present.

For example, in an InstallScript event handler such as OnFirstUIBefore or in an InstallScript custom action, you can use code such as the following snippet:

```
if (SYSINFO.bIsVirtualMachine) then
    SprintfBox (0, "VM", "This is a virtual machine. Type: %d",
        GetSystemInfo (VIRTUAL_MACHINE_TYPE, nNumber, szString));
else
    MessageBox ("This is not a virtual machine.", 0);
```

```
endif;
```

At run time, a message box is displayed. If the target system is a Microsoft Hyper-V image, a VMware image, or Microsoft Virtual PC, the message box says, "This is a virtual machine," and it also lists the type of virtual machine. If the target system is not one of those virtual machines, the message box says, "This is not a virtual machine."

Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where appropriate.



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

How Advanced UI and Suite/Advanced UI Installations Use Conditions

The Advanced UI and Suite/Advanced UI project types in InstallShield have support for building conditional statements that control different areas of the run time:

- **Exit conditions**—The Exit Condition setting in the General Information view lets you specify one or more exit error messages that you want the Advanced UI or Suite/Advanced UI installation to display under various conditions before ending the installation. If the condition that you define is true when an end user launches the Advanced UI or Suite/Advanced UI installation, the Advanced UI or Suite/Advanced UI installation displays the error message. When end users dismiss the error message, the Advanced UI or Suite/Advanced UI installation ends.
- **Detection conditions**—The Detection Condition setting for a package in the Packages view lets you specify the condition that the Advanced UI or Suite/Advanced UI installation should use to evaluate whether the package is already installed on target systems. If this does not change from false to true (or from true to false) at run time, the Advanced UI or Suite/Advanced UI installation assumes that the package failed to install (or remove) its payload. In addition, subsequent maintenance and remove operations (or subsequent maintenance and install operations) may not behave as expected.
- **Eligibility conditions**—The Eligibility Condition setting for a package in the Packages view lets you specify the condition that the Advanced UI or Suite/Advanced UI installation should use to determine whether the target system meets the requirements that are necessary for the package to be run. For example, if the package should be run only on 64-bit systems, you could set up an x64 platform requirement in a condition; the Advanced UI or Suite/Advanced UI installation would launch the package only on 64-bit systems. You may also want to set up an eligibility condition to prevent end users from being able to install the current package version over a future newer version.

- **Feature conditions**—The Condition setting in the Features view lets you build conditional statements to evaluate whether the feature should be selected for installation by default on the InstallationFeatures wizard page.
- **Action conditions**—The Condition setting for an action that is scheduled for an event in the Events view lets you build conditional statements that evaluate whether the installation should run the action.

  In addition, the settings in the Events area for a package in the Packages view let you build conditional statements that evaluate whether the installation should run an action for the selected package.

  Project • *Action conditions are available in Suite/Advanced UI projects.*
- **Wizard interface conditions**—The condition settings for wizard pages and controls in the Wizard Interface view let you build conditional statements that evaluate behavior such as whether the installation should invoke an action, validate an end user's response, enable a control, show a page or control.
- **Mode conditions**—InstallShield creates two types of mode conditions automatically: install mode conditions and maintenance mode conditions. The mode conditions are shown only in the project file (.issuite); this type of condition is not available for edit within InstallShield. These mode conditions determine whether an Advanced UI or Suite/Advanced UI installation runs in first-time installation mode or in maintenance mode. For more details, see Triggering Install Mode or Maintenance Mode of an Advanced UI or Suite/Advanced UI Installation.

Condition Operators

Each condition that you define in an Advanced UI or Suite/Advanced UI project begins with one of the following operators:

- **Any**—An Any condition group operates like a logical OR operation. If any of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to true. If none of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to false.
- **All**—An All condition group operates like a logical AND operation. All of the conditions that are in the All group must evaluate to true in order for the condition group to evaluate to true.
- **None**—A None condition group operates like a logical NOR operation. If none of the conditions that are in the None group evaluate to true, the entire condition group evaluates to true. If any of the conditions that are in the None condition group evaluate to true, the entire condition group evaluates to false. Note that if the None condition group consists of only one conditional statement, it operates like a logical NOT operation.

Types of Condition Checks

When you are building a conditional statement for an exit, detection, eligibility, feature, or wizard interface condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can select from a number of different types of checks that you want to be evaluated on target systems. Examples are operating system version, presence of a file or registry entry, and presence of an already installed product.

For detailed information about each of the types of condition checks that are supported, as well as each of the subsettings that are displayed under a condition, see Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects.

Syntax for Building Conditions

Building a conditional statement involves using one or more of the condition settings to add operators and condition checks.

To add a conditional statement to a condition setting, click the setting's New Condition button:



When you do that, InstallShield adds an operator row under the condition setting; the default operator is Any. To change the operator for the condition, click the drop-down arrow in the operator setting and then click the appropriate option.

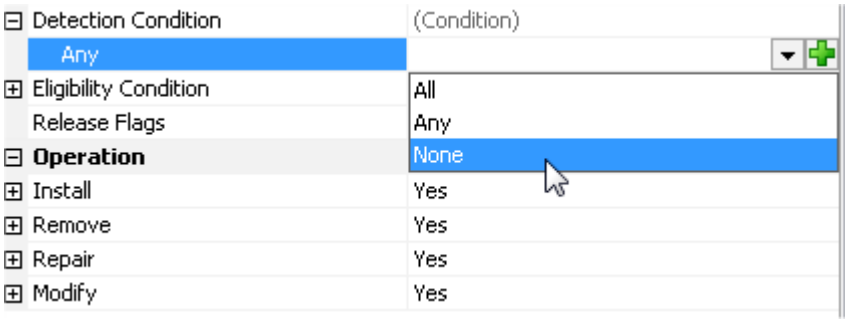


Figure 9-1: Changing the Default Operator

Once you have added an operator, click the New Condition button in the operator setting and then click the appropriate condition check type. At this point, InstallShield adds a subsetting (or more subsettings, depending on which condition check type you selected) under the operator setting.

Whenever a conditional statement is invalid (for example, if an entry is required in a setting but it is left blank), InstallShield shows a warning icon in that setting. To find out why the condition is invalid, move the pointer over an icon; InstallShield displays a tooltip that explains the issue.



To nest another condition within a condition, click the New Condition button in an operator setting and then click the appropriate operator for the nested condition. One of the Any/All/None meta-conditions is referred to as a *condition group*. All of the nested condition groups and conditions under a condition setting together form a *condition tree*.

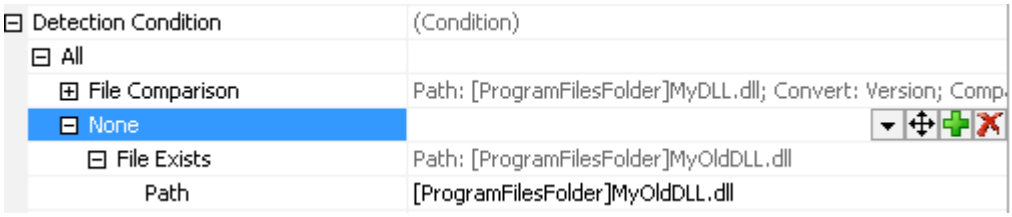


Figure 9-2: Nesting a Condition Within a Condition

To move a condition or a condition group within a condition tree, point to the Move Condition button in the setting of the item that you want to move, and then click the appropriate direction (Move Up, Move Down, Move Left, or Move Right).

You can use an asterisk as a placeholder for the current package's product code, package code, and product version when you are defining eligibility and detection MSI package conditions, as long as the package that you are configuring is an .msi package.



Figure 9-3: Default Eligibility Condition for an .msi Package

You can also use an asterisk as a placeholder for the current package's upgrade code and either the minimum or maximum version number when you are defining eligibility and detection MSI upgrade conditions, as long as the package that you are configuring is an .msi package.

Types of Condition Checks in Advanced UI and Suite/Advanced UI Projects

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, feature, or wizard interface condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can select from a number of different types of checks that you want to be evaluated on target systems.

Each type of condition has one or more subsettings that are associated with it. The following types of conditions are available:

Table 9-8 • Types of Condition Checks

Type of Condition Check	Description
Platform	Check target systems for one or more characteristics such as the operating system version, the architecture, and other platform details on target systems. For descriptions of each of the settings that are displayed for this type of condition check, see Platform Condition Settings.

Table 9-8 • Types of Condition Checks (cont.)

Type of Condition Check	Description
File Exists	Check target systems for the presence of a particular file. For the description of the setting that is displayed for this type of condition check, see File Exists Condition Settings.
File Comparison	Check target systems for specific information—date, version number, or content—for a particular file. For descriptions of each of the settings that are displayed for this type of condition check, see File Comparison Condition Settings.
Registry Exists	Check target systems for the presence of a particular registry key, and optionally a specific value name. For descriptions of each of the settings that are displayed for this type of condition check, see Registry Exists Condition Settings.
Registry Comparison	Check target systems for specific information—DWORD, QWORD, string, multistring, file version, or product version—for a particular registry entry. If you do not specify a value name, the default value for the key that you specify is used for the comparison. For descriptions of each of the settings that are displayed for this type of condition check, see Registry Comparison Condition Settings.
Property Comparison	Check the value of a particular built-in Advanced UI or Suite/Advanced UI property, or a property that is defined in the Property Manager view. For descriptions of each of the settings that are displayed for this type of condition check, see Property Comparison Condition Settings.
MSI Package	Check target systems for the presence of a product that was installed by a particular .msi package. For descriptions of each of the settings that are displayed for this type of condition check, see MSI Package Condition Settings.
MSI Upgrade	Check target systems for the presence of a version of a product that you want to update, or for the presence of future versions of a product that you do not want to downgrade. In either case, the product whose presence is being checked is installed by an .msi package. For descriptions of each of the settings that are displayed for this type of condition check, see MSI Upgrade Condition Settings.

Table 9-8 • Types of Condition Checks (cont.)

Type of Condition Check	Description
Eligible Package	Check whether a different package or a feature in the Advanced UI or Suite/Advanced UI installation is eligible for installation on a target system. For example, if a patch package is eligible for installation only if the base package is eligible for installation, you can create an Eligible Package condition to tie the patch's eligibility to the base package's eligibility. In this scenario, you would select the patch package in the Packages view, create the condition, and specify the package GUID of the base package in that condition. For a description of the setting that is displayed for this type of condition check, see Eligible Package Condition Setting.
InstallScript Package	Check target systems for the presence of a product that was installed by a particular InstallScript package. For descriptions of each of the settings that are displayed for this type of condition check, see InstallScript Package Condition Settings.
Locale	Match one or more locale-related settings on target systems. For descriptions of each of the settings that are displayed for this type of condition check, see Locale Condition Settings.
Suite Installed	Check whether a particular version of an Advanced UI or Suite/Advanced UI installation is already installed on a target system. For descriptions of each of the settings that are displayed for this type of condition check, see Suite Installed Condition Settings.
UWP App Package	Check target systems for the presence of a particular UWP app. For descriptions of each of the settings that are displayed for this type of condition check, see UWP App Package Condition Settings.
UWP App Package Eligible	Check target systems for the run-time dependencies of the UWP app package to prevent any attempts to install a UWP app package onto a version of Windows or Windows Server that does not support it. This condition is available only for the eligibility condition of a UWP app package. If it is used in any other package type, it will not function correctly.  ***Tip*** • *Sideloading must be enabled for a UWP app package to be successfully sideloaded. Sideloading works on WIndows 10 or newer. For more information about sideloading, refer to the Enable your device for deployment MSDN article.*

Table 9-8 • Types of Condition Checks (cont.)

Type of Condition Check	Description
UWP Type Present	Check target systems for the presence of UWP functionality. For example, to create a conditional statement that checks for the presence of the Desktop Bridge, check for the type `Windows.ApplicationModel.FullTrustProcessLauncher`. This can be used to conditionally block installation, or to choose between installing .msi and UWP app package (.appx). For descriptions of each of the setting that is displayed for this type of condition check, see UWP Type Present Condition Settings.
Package Operation	 ***Project*** • *This type of condition is available in Suite/Advanced UI projects.* Check the target state of a particular package in the Suite/Advanced UI installation. This type of condition is available only for actions that are associated with an event in the Events view or Packages view. For descriptions of each of the settings that are displayed for this type of condition check, see Package Operation Condition Settings.
Feature Operation	 ***Project*** • *This type of condition is available in Suite/Advanced UI projects.* Check the target state of a particular feature in the Suite/Advanced UI installation. This type of condition is available only for actions that are associated with an event in the Events view or Packages view. For descriptions of each of the settings that are displayed for this type of condition check, see Feature Operation Condition Settings.
Extension Condition DLL	Use a C/C++ DLL that you created to implement your own custom condition that checks target systems for factors that the other types of condition checks do not evaluate. For descriptions of each of the settings that are displayed for this type of condition check, see Extension Condition Settings.
Mode	Check the mode in which the Advanced UI or Suite/Advanced UI installation is running. This type of condition check is available for various settings of wizard interface elements in the Wizard Interface view. For descriptions of each of the settings that are displayed for this type of condition check, see Mode Condition Settings.

Platform Condition Settings



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a platform condition.

The Platform setting shows the platform-related conditions that are configured in the Platform subsettings. To define a condition that evaluates the operating system version, the architecture, and other platform details on target systems, configure one or more of the platform-related subsettings as needed.

The following subsettings are available for this condition.

Table 9-9 • Platform Condition Settings

Setting	Description
OS Version	To create a conditional statement that evaluates the version of Windows on target systems, do one of the following: • To check for an exact match, enter the version number of Windows in the format *major.minor*. • To specify the minimum and maximum version numbers, enter a range separated by a dash. For example, to test for version 5.0 through 6.1 of Windows, enter the following: **5.0-6.1** • To check for a minimum version number, enter the minimum version number followed by a dash. For example, to test for version 5.0 and later versions, enter the following: **5.0-** • To check for a maximum version number, enter a dash followed by the maximum version number. For example, to test for version 6.0 and earlier versions, enter the following: **-6.0** For a list of valid versions, see the dwMajorVersion and dwMinorVersion members of the OSVERSIONINFOEX structure that is documented on the MSDN Web site.  ***Important*** • *If you specify a range of operating system versions in the OS Version setting, as well as a range of service pack numbers in the Service Pack setting, ensure that the conditions evaluate as expected at run time.*

Table 9-9 • Platform Condition Settings (cont.)

Setting	Description
Service Pack	To create a conditional statement that evaluates the service pack number of Windows on target systems, do one of the following: • To check for an exact match, enter the specific service pack number of Windows. For example, to test for SP2, enter the number **2**. • To specify the minimum and maximum service pack numbers, enter a range separated by a dash. For example, to test for SP1 through SP3 of Windows, enter the following: **1-3** • To check for a minimum service pack number, enter the minimum service pack number followed by a dash. For example, to test for SP2 and later versions, enter the following: **2-** • To check for a maximum service pack number, enter a dash followed by the maximum service pack number. For example, to test for SP2 and earlier versions, enter the following: **-2** For a list of valid service pack numbers, see the wServicePackMajor and wServicePackMinor members of the OSVERSIONINFOEX structure that is documented on the MSDN Web site.  ***Important*** • *If you specify a range of operating system versions in the OS Version setting, as well as a range of service pack numbers in the Service Pack setting, ensure that the conditions evaluate as expected at run time.*
CSD Version	To create a conditional statement that evaluates the CSD version of Windows on target systems, enter the specific CSD version. **Service Pack 2** is an example of a CSD version.
Build Number	To create a conditional statement that evaluates the minimum build number version of Windows on target systems, enter the specific build number. For a list of valid build numbers, see the dwBuildNumber member OSVERSIONINFOEX structure that is documented on the MSDN Web site.
Product Type	To create a conditional statement that evaluates whether the target system is a domain controller, a server, or a workstation, select the appropriate option. For a list of valid types, see the wProductType member OSVERSIONINFOEX structure that is documented on the MSDN Web site.

Table 9-9 • Platform Condition Settings (cont.)

Setting	Description
Architecture	To create a conditional statement that evaluates the architecture (x86, x64, or IA64) of target systems, select the appropriate option.

File Exists Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a File Exists condition.

The File Exists setting shows the condition that is configured for checking target systems for the presence of a particular file. To define this type of condition, configure the subsettings under the File Exists setting as needed.

The following subsettings are available for this condition.

Table 9-10 • File Exists Condition Settings

Setting	Description
Path	To create a conditional statement that checks for the presence of a particular file, do one of the following: • Enter the path and name for the file. To include a predefined folder in the path, select the appropriate property in the list. If you select one of the 64-bit folder locations, the installation checks the 64-bit folder location on 64-bit target systems and the 32-bit folder location on 32-bit target systems. • Enter only the file name in this Path setting; use the Search Path setting to specify one or more paths that you want the installation to check at run time. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

Table 9-10 • File Exists Condition Settings (cont.)

Setting	Description
Search Path	If you specified a path and file name in the Path setting, leave the Search Path setting blank. If you specified only a file name in the Path setting, use the Search Path setting to specify a list of the paths for the locations on target systems in which you want to check for that file. Separate multiple paths with a semicolon (;). To include a predefined folder in the path, use the appropriate property. The following sample value for the Search Path setting uses directory properties to search two different paths: **[CommonFilesFolder]Microsoft Shared\Office15;[CommonFilesFolder]Microsoft Shared\Office14** It is strongly recommended that you try to narrow the search path as much as possible. In general, the more broad you make your search path, the longer it will take for the installation to check the target system. Note that if any path that you enter in this setting includes the property [System64Folder], the installation disables file system redirection on 64-bit target systems when checking the system folder for any of the specified paths. If you do not enter [System64Folder] in this setting for any paths, the installation leaves file system redirection enabled. Thus, if you enter a value of **[System64Folder];[SystemFolder]** in this setting, the installation checks **C:\Windows\System32** on both 32-bit and 64-bit target systems. If you enter a value of **[SystemFolder]**, the installation checks **C:\Windows\System32** on 32-bit target systems and **C:\Windows\SysWOW64** on 64-bit target systems. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. For example, to search for the file that is specified in the Path setting in all directories that are defined for the PATH environment variable on a target system, enter the following value in the Search Path setting: **[%PATH]**

File Comparison Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a File Comparison condition.

The File Comparison setting shows the condition that is configured for checking target systems for specific information—date, version number, or content—for a particular file. To define this type of condition, configure the subsettings under the File Comparison setting as needed.

The following subsettings are available for this condition.

Table 9-11 • File Comparison Condition Settings

Setting	Description
Path	To create a conditional statement that checks for specific information for a particular file, do one of the following: • Enter the path and name for the file. To include a predefined folder in the path, select the appropriate property in the list. If you select one of the 64-bit folder locations, the installation checks the 64-bit folder location on 64-bit target systems and the 32-bit folder location on 32-bit target systems. • Enter only the file name in this Path setting; use the Search Path setting to specify one or more paths that you want the installation to check at run time. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

Table 9-11 • File Comparison Condition Settings (cont.)

Setting	Description
Search Path	If you specified a path and file name in the Path setting, leave the Search Path setting blank. If you specified only a file name in the Path setting, use the Search Path setting to specify a list of the paths for the locations on target systems in which you want to check for that file. Separate multiple paths with a semicolon (;). To include a predefined folder in the path, use the appropriate property. The following sample value for the Search Path setting uses directory properties to search two different paths: **[CommonFilesFolder]Microsoft Shared\Office15;[CommonFilesFolder]Microsoft Shared\Office14** It is strongly recommended that you try to narrow the search path as much as possible. In general, the more broad you make your search path, the longer it will take for the installation to check the target system. Note that if any path that you enter in this setting includes the property [System64Folder], the installation disables file system redirection on 64-bit target systems when checking the system folder for any of the specified paths. If you do not enter [System64Folder] in this setting for any paths, the installation leaves file system redirection enabled. Thus, if you enter a value of **[System64Folder];[SystemFolder]** in this setting, the installation checks **C:\Windows\System32** on both 32-bit and 64-bit target systems. If you enter a value of **[SystemFolder]**, the installation checks **C:\Windows\System32** on 32-bit target systems and **C:\Windows\SysWOW64** on 64-bit target systems. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. For example, to search for the file that is specified in the Path setting in all directories that are defined for the PATH environment variable on a target system, enter the following value in the Search Path setting: **[%PATH]**
Conversion	Select the type of data that you want to compare. Available options are: • **Content**—The data that you want to compare is a string. • **Version**—The data that you want to compare is a file version in the format *nn.nn.nn.nn*. • **Local Date and Time**—The data that you want to compare is the local date and time on the target system that is running the Advanced UI or Suite/Advanced UI installation—for example, **4/26/2004 3:57:46 PM**.

Table 9-11 • File Comparison Condition Settings (cont.)

Setting	Description
Comparison	Select the option that describes how you want to compare the value on the target system with the value that you specify in the Compare To setting. For example, consider a file version condition in which the value of the Comparison setting is Less Than or Equal To, and the value of the Compare To setting is 4.1. At run time, this condition evaluates as true if the file version on the target system is 4.0 or 4.1; it evaluates as false if the file version on the target system is 4.2.
Compare To	Specify the value that you want to compare with the value on the target system. The value that you enter depends on the option that is selected for the Conversion setting: • If Content is selected in the Conversion setting, enter the file string that you want to compare in the Compare To setting. • If Version is selected in the Conversion setting, enter the file version in the format *nn.nn.nn.nn*. • If Local Date and Time is selected in the Conversion setting, enter the date and time that is used locally on target systems that are running the Advanced UI or Suite/Advanced UI installation—for example, **4/26/2004 3:57:46 PM**. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. The Comparison setting is where you indicate how you want to compare the value on the target system with the value that you specify in the Compare To setting. For example, consider a file version condition in which the value of the Comparison setting is Less Than or Equal To, and the value of the Compare To setting is 4.1. At run time, this condition evaluates as true if the file version on the target system is 4.0 or 4.1; it evaluates as false if the file version on the target system is 4.2.

Registry Exists Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a Registry Exists condition.

The Registry Exists setting shows the condition that is configured for checking target systems for the presence of a particular registry key, and optionally a specific value name. To define this type of condition, configure the subsettings under the Registry Exists setting as needed.

The following subsettings are available for this condition.

Table 9-12 • Registry Exists Condition Settings

Setting	Description
Registry Key	To create a conditional statement that checks for the presence of a particular registry key, select the appropriate root key in this setting, and then enter the rest of the registry path. Available root keys are: • **HKLM**—HKEY_LOCAL_MACHINE • **HKCU**—HKEY_CURRENT_USER • **HKCR**—HKEY_CLASSES_ROOT • **HKU**—HKEY_USERS When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
Value Name	To create a conditional statement that checks for the presence of a particular registry value for the specified registry key, enter the appropriate registry value. If you do not want to check for the presence of a specific registry value—that is, you want to check only for the presence of a registry key—leave this setting blank. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

Table 9-12 • Registry Exists Condition Settings (cont.)

Setting	Description
64-Bit Key	Specify whether you want the Advanced UI or Suite/Advanced UI installation to check the 64-bit portion of the registry on 64-bit systems. If you select True, the Advanced UI or Suite/Advanced UI installation ignores this conditional statement on 32-bit target systems.

Registry Comparison Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a Registry Comparison condition.

The Registry Comparison setting shows the condition that is configured for checking target systems for specific information—DWORD, QWORD, string, multistring, file version, or product version—for a particular registry entry. If you do not specify a value name, the default value for the key that you specify is used for the comparison. To define this type of condition, configure the subsettings under the Registry Comparison setting as needed.

The following subsettings are available for this condition.

Table 9-13 • Registry Comparison Condition Settings

Setting	Description
Registry Key	To create a conditional statement that checks for specific information for a particular value name of a registry key, select the appropriate root key in this setting, and then enter the rest of the registry path. Available root keys are: • **HKLM**—HKEY_LOCAL_MACHINE • **HKCU**—HKEY_CURRENT_USER • **HKCR**—HKEY_CLASSES_ROOT • **HKU**—HKEY_USERS
Value Name	Enter the registry value that you want to check for the registry comparison condition. If you want to check the registry key's default value, leave this setting blank.

Table 9-13 • Registry Comparison Condition Settings (cont.)

Setting	Description
64-Bit Key	Specify whether you want the Advanced UI or Suite/Advanced UI installation to check the 64-bit portion of the registry on 64-bit systems. If you select True, the Advanced UI or Suite/Advanced UI installation ignores this conditional statement on 32-bit target systems.
Conversion	Select the type of value that you want to compare. Available options are: • **DWORD**—The value that you want to compare is a DWORD (32-bit) value. • **QWORD**—The value that you want to compare is a QWORD (64-bit) value. • **String**—The value that you want to compare is a string value. • **Multistring**—The value that you want to compare is a multistring value. • **File Version**—The value that you want to compare is a file version value in the format *nn.nn.nn.nn*. • **Product Version**—The value that you want to compare is a product version value. The typical format for the version number is *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. Note that although you can include the fourth field (*ddddd*) when you specify the product version, the Advanced UI or Suite/Advanced UI installation does not use this part of the product version to distinguish between different product versions.
Comparison	Select the option that describes how you want to compare the value on the target system with the value that you specify in the Compare To setting. For example, consider a registry condition in which the value of the Comparison setting is Less Than or Equal To, and the value of the Compare To setting is 4.1. At run time, this condition evaluates as true if the number in the specified registry value data on the target system is 4.0 or 4.1; it evaluates as false if the version on the target system is 4.2.

Table 9-13 • Registry Comparison Condition Settings (cont.)

Setting	Description
Compare To	Specify the value that you want to compare with the value on the target system. The value that you enter depends on the option that is selected for the Conversion setting: • If DWORD is selected in the Conversion setting, enter the DWORD value that you want to compare in the Compare To setting. • If QWORD is selected in the Conversion setting, enter the QWORD value that you want to compare in the Compare To setting. • If String is selected in the Conversion setting, enter the string value that you want to compare in the Compare To setting. • If MultiString is selected in the Conversion setting, enter the multistring value that you want to compare in the Compare To setting. • If File Version is selected in the Conversion setting, enter the file version value (in the format *nn.nn.nn.nn*) that you want to compare in the Compare To setting.
Compare To (continued)	• If Product Version is selected in the Conversion setting, enter the product version value that you want to compare in the Compare To setting. The typical format for the version number is *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. Note that although you can include the fourth field (*ddddd*) when you specify the product version, the Advanced UI or Suite/Advanced UI installation does not use this part of the product version to distinguish between different product versions. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. The Comparison setting is where you indicate how you want to compare the value on the target system with the value that you specify in the Compare To setting. For example, consider a registry condition in which the value of the Comparison setting is Less Than or Equal To, and the value of the Compare To setting is 4.1. At run time, this condition evaluates as true if the number in the specified registry value data on the target system is 4.0 or 4.1; it evaluates as false if the version on the target system is 4.2.

Property Comparison Condition Settings

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, feature, or wizard interface condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a Property Comparison condition. The Property Comparison setting shows the condition that is configured for checking the value of a particular built-in Advanced UI or Suite/Advanced UI property, or a property that is defined in the Property Manager view. To define this type of condition, configure the subsettings under the Property Comparison setting as needed.

The following subsettings are available for this condition.

Table 9-14 • Property Comparison Condition Settings

Setting	Description
Name	To create a conditional statement that checks the value of a particular Advanced UI or Suite/Advanced UI property, enter the name of the property in this setting. For a list of built-in conditions, see Advanced UI and Suite/Advanced UI Property Reference.

Table 9-14 • Property Comparison Condition Settings (cont.)

Setting	Description
Conversion	Select the type of value that you want to compare. Available options are: • **DWORD**—The value that you want to compare is a DWORD (32-bit) value. • **QWORD**—The value that you want to compare is a QWORD (64-bit) value. • **String**—The value that you want to compare is a string value. • **File Version**—The value that you want to compare is a file version value in the format *nn.nn.nn.nn*. • **Product Version**—The value that you want to compare is a product version value. The typical format for the version number is *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. Note that although you can include the fourth field (*ddddd*) when you specify the product version, the Advanced UI or Suite/Advanced UI installation does not use this part of the product version to distinguish between different product versions. • **Local Date Time**—The value that you want to compare is a date or a date and time.
Comparison	Select the option that describes how you want to compare the value of the property at run time on the target system with the value that you specify in the Compare To setting. For example, consider a property comparison condition in which the value of the Conversion setting is String, the value of the Comparison setting is Equal, and the value of the Compare To setting is MyValue. At run time, this condition evaluates as true if the property value at run time is MyValue; it evaluates as false if the property value is not set or is any other value.

Table 9-14 • Property Comparison Condition Settings (cont.)

Setting	Description
Compare To	Specify the value that you want to compare with the value of the Advanced UI or Suite/Advanced UI property on the target system at run time. The value that you enter depends on the option that is selected for the Conversion setting: • If DWORD is selected in the Conversion setting, enter the DWORD value that you want to compare in the Compare To setting. • If QWORD is selected in the Conversion setting, enter the QWORD value that you want to compare in the Compare To setting. • If String is selected in the Conversion setting, enter the string value that you want to compare in the Compare To setting. • If File Version is selected in the Conversion setting, enter the file version value (in the format *nn.nn.nn.nn*) that you want to compare in the Compare To setting.
Compare To (continued)	• If Product Version is selected in the Conversion setting, enter the product version value that you want to compare in the Compare To setting. The typical format for the version number is *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. Note that although you can include the fourth field (*ddddd*) when you specify the product version, the Advanced UI or Suite/Advanced UI installation does not use this part of the product version to distinguish between different product versions. • If Local Date Time is selected in the Conversion setting, enter the date and time that you want to compare in the Compare To setting—for example, **4/26/2004 3:57:46 PM**. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. The Comparison setting is where you indicate how you want to compare the value of the property at run time on the target system with the value that you specify in the Compare To setting. For example, consider a property comparison condition in which the value of the Conversion setting is String, the value of the Comparison setting is Equal, and the value of the Compare To setting is MyValue. At run time, this condition evaluates as true if the property value at run time is MyValue; it evaluates as false if the property value is not set or is any other value.

MSI Package Condition Settings

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create an MSI Package condition.

The MSI Package setting shows the condition that is configured for checking target systems for the presence of a product that was installed by a particular .msi package. To define this type of condition, configure the subsettings under the MSI Package setting as needed.

The following subsettings are available for this condition.

Table 9-15 • MSI Package Condition Settings

Setting	Description
Product Code	To create a conditional statement that checks for the presence of a product that was installed by an .msi package with a particular product code, click the ellipsis button (...) in this setting to browse to the package and have InstallShield enter the product code for you. As an alternative, you can manually enter the product code. A sample entry is {5D607F6A-AF48-4003-AFA8-69E019A4496F}. ***Tip*** *• You can use an asterisk as a placeholder for the current package's product code, package code, and product version when you are defining eligibility and detection MSI package conditions, as long as the package that you are configuring is an .msi package.*

Table 9-15 • MSI Package Condition Settings (cont.)

Setting	Description
Package Code	To create a conditional statement that checks for the presence of a product that was installed by an .msi package with a particular package code, click the ellipsis button (...) in this setting to browse to the package and have InstallShield enter the package code for you. As an alternative, you can manually enter the package code. A sample entry is {5D607F6A-AF48-4003-AFA8-69E019A4496F}. ***Tip*** • *You can use an asterisk as a placeholder for the current package's product code, package code, and product version when you are defining eligibility and detection MSI package conditions, as long as the package that you are configuring is an .msi package.*
Product Version	To create a conditional statement that checks for the presence of a product with a particular product version that was installed by an .msi package, click the ellipsis button (...) in this setting to browse to the package and have InstallShield enter the product version for you. As an alternative, you can manually enter the product version. Then use the Comparison setting to specify whether you want the product version to be less than, greater than, or equal to the value that you are specifying. The typical format for the version number is *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the aaa and bbb portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535. Note that although you can include the fourth field (*ddddd*) when you specify the product version, the Advanced UI or Suite/Advanced UI installation does not use this part of the product version to distinguish between different product versions. ***Tip*** • *You can use an asterisk as a placeholder for the current package's product code, package code, and product version when you are defining eligibility and detection MSI package conditions, as long as the package that you are configuring is an .msi package.*
Comparison	If you specify a version number in the Product Version setting, select the appropriate option to indicate how you want to compare the version number on the target system with the one that you enter. For example, if you want the condition to be true if the version on the target system is less than or equal to 4.0.0, enter 4.0.0 in the Product Version setting, and select Less Than or Equal To in the Comparison setting.

Table 9-15 • MSI Package Condition Settings (cont.)

Setting	Description
Patch Code	To create a conditional statement that checks for the presence of a product that was installed by an .msi package with a particular patch code for a specific product code, enter the patch code in this setting, and enter the product code in the Product Code setting. A sample entry is {5D607F6A-AF48-4003-AFA8-69E019A4496F}. If you enter a value in this setting, any value that is entered in the Package Code and Product Version settings are ignored.

MSI Upgrade Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create an MSI Upgrade condition.

The MSI Upgrade setting shows the condition that is configured for checking target systems for the presence of a version of a product that you want to update, or for the presence of future versions of a product that you do not want to downgrade. In either case, the product whose presence is being checked is installed by an .msi package. To define this type of condition, configure the subsettings under the MSI Upgrade setting as needed.

The following subsettings are available for this condition.

Table 9-16 • MSI Upgrade Condition Settings

Setting	Description
Upgrade Code	To create a conditional statement that checks for the presence of a product that was installed by an .msi package with a particular upgrade code, click the ellipsis button (...) in this setting to browse to the package and have InstallShield enter the upgrade code for you. As an alternative, you can manually enter the upgrade code. A sample entry is {5D607F6A-AF48-4003-AFA8-69E019A4496F}. ***Tip*** *• You can use an asterisk (*) as a placeholder for the current package's upgrade code and either the minimum or maximum version number when you are defining eligibility and detection MSI upgrade conditions, as long as the package that you are configuring is an .msi package.*
Minimum Version	Enter the minimum product version that the conditional statement should evaluate when determining whether to run the upgrade package, or click the ellipsis button (...) in this setting to browse to the package and have InstallShield enter the minimum version. Version numbers should be in the format xxxx.xxxx.xxxx. ***Tip*** *• You can use an asterisk (*) as a placeholder for the current package's upgrade code and either the minimum or maximum version number when you are defining eligibility and detection MSI upgrade conditions, as long as the package that you are configuring is an .msi package.*
Include Minimum Version	Indicate whether you want to include the minimum version value in the range of product versions that you want to include in the conditional statement. This setting helps you specify whether you are using the lower bounds for a range of product versions that your upgrade package will or will not support.
Maximum Version	Enter the maximum product version that the conditional statement should evaluate when determining whether to run the upgrade package, or click the ellipsis button (...) in this setting to browse to the package and have InstallShield enter the maximum version. Version numbers should be in the format xxxx.xxxx.xxxx. ***Tip*** *• You can use an asterisk as a placeholder for the current package's upgrade code and either the minimum or maximum version number when you are defining eligibility and detection MSI upgrade conditions, as long as the package that you are configuring is an .msi package.*

Table 9-16 • MSI Upgrade Condition Settings (cont.)

Setting	Description
Include Maximum Version	Indicate whether you want to include the maximum version value in the range of product versions that you want to include in the conditional statement. This setting helps you specify whether you are using the upper bounds for a range of product versions that your upgrade package will or will not support.

Eligible Package Condition Setting



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create an Eligible Package condition.

The Eligible Package setting shows the condition that is configured for checking whether a different package or a feature in the Advanced UI or Suite/Advanced UI installation is eligible for installation on a target system. For example, if a patch package is eligible for installation only if the base package is eligible for installation, you can create an Eligible Package condition to tie the patch's eligibility to the base package's eligibility. In this scenario, you would select the patch package in the Packages view, create the condition, and specify the package GUID of the base package in that condition.

To define this type of condition, configure the subsetting under the Eligible Package setting as needed.

The following subsetting is available for this condition.

Table 9-17 • Eligible Package Condition Setting

Setting	Description
Package GUID	To create a conditional statement that is based on whether a different package in the Advanced UI or Suite/Advanced UI installation is eligible for installation on a target system, select the other required package in the list in this setting. The list contains all of the packages in the project. As an alternative, you can enter the package GUID of that other required package. A sample entry is {5D607F6A-AF48-4003-AFA8-69E019A4496F}. For example, if a patch package is eligible for installation only if the base product is eligible for installation, you can create an Eligible Package condition to tie the patch's eligibility to the base product's eligibility. In this scenario, you would select the patch package in the Packages view, create the condition, and select the base package in that condition. The package GUID is identified in the Package GUID setting in the Packages view.

InstallScript Package Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create an InstallScript Package condition.

The InstallScript Package setting shows the condition that is configured for checking target systems for the presence of a product that was installed by a particular InstallScript package. To define this type of condition, configure the subsettings under the InstallScript Package setting as needed.

The following subsettings are available for this condition.

Table 9-18 • InstallScript Package Condition Settings

Setting	Description
Product Code	To create a conditional statement that checks for the presence of a product that was installed by a particular InstallScript package with a particular product code, click the ellipsis button (...) in this setting to browse to the **data1.hdr** file of the uncompressed InstallScript package and have InstallShield enter the product code for you. As an alternative, you can manually enter the product code. A sample entry is {5D607F6A-AF48-4003-AFA8-69E019A4496F}.
Product Version	To create a conditional statement that checks for the presence of a product with a particular product version that was installed by an InstallScript package, enter the product version. Then use the Comparison setting to specify whether you want the product version to be less than, greater than, or equal to the value that you are specifying. The typical format for the version number is *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the aaa and bbb portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535. Note that although you can include the fourth field (*ddddd*) when you specify the product version, the Advanced UI or Suite/Advanced UI installation does not use this part of the product version to distinguish between different product versions.
Verify Log Exists	Select the option that describes whether you want the Advanced UI or Suite/Advanced UI installation to verify that the uninstallation log file that is registered for the product is present on target systems. If you leave this setting blank, the Advanced UI or Suite/Advanced UI installation behaves as if you selected Yes in this setting.
Check Per-User Registration	Select the option that describes whether you want the Advanced UI or Suite/Advanced UI installation to check HKEY_CURRENT_USER for uninstallation information on target systems. Available options are: • **Yes**—The installation checks both HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE for uninstallation information. • **No**—The installation checks only HKEY_LOCAL_MACHINE for uninstallation information. If you leave this setting blank, the Advanced UI or Suite/Advanced UI installation behaves as if you selected Yes in this setting.

Table 9-18 • InstallScript Package Condition Settings (cont.)

Setting	Description
Comparison	If you specify a version number in the Product Version setting, select the appropriate option to indicate how you want to compare the version number on the target system with the one that you enter. For example, if you want the condition to be true if the version on the target system is less than or equal to 4.0.0, enter 4.0.0 in the Product Version setting, and select Less Than or Equal To in the Comparison setting.

Locale Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a Locale condition.

The Locale setting shows the condition that is configured for matching one or more locale-related settings on target systems. To define this type of condition, configure one or more of the subsettings under the Locale setting as needed.

The following subsettings are available for this condition.

Table 9-19 • Locale Condition Settings

Setting	Description
User Interface	To create a conditional statement that checks for the locale of the user interface on target systems, enter the locale identifier (LCID) in any of the following formats: • Decimal LCID; for example: **1033** • Hexadecimal LCID preceded with 0x; for example: **0x409** • Locale name, which is checked only on Windows Vista and later systems, and ignored on earlier systems; for example: **en-us**, **en**, **zh-CN** For a list of LCIDs, see Locale IDs Assigned by Microsoft on MSDN. For more details on entering values in this setting, see Using Locale Conditions in Advanced UI and Suite/Advanced UI Projects.
Operating System	To create a conditional statement that checks for the language of the operating system on target systems, enter the locale identifier (LCID) in either of the following formats: • Decimal LCID; for example: **1033** • Hexadecimal LCID preceded with 0x; for example: **0x409** For a list of LCIDs, see Locale IDs Assigned by Microsoft on MSDN. For more details on entering values in this setting, see Using Locale Conditions in Advanced UI and Suite/Advanced UI Projects.
ANSI Code Page	To create a conditional statement that checks for a ANSI code page on target systems, enter the code page identifier. For a list of code page identifiers, see Code Page Identifiers on MSDN. For more details on entering values in this setting, see Using Locale Conditions in Advanced UI and Suite/Advanced UI Projects.

Suite Installed Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a Suite Installed condition.

The Suite Installed setting shows the condition that is configured for checking whether a particular version of an Advanced UI or Suite/Advanced UI installation is already installed on a target system. To define this type of condition, configure one or more of the subsettings under the Suite Installed setting as needed.

The following subsettings are available for this condition.

Table 9-20 • Suite Installed Condition Settings

Setting	Description
Suite GUID	To create a conditional statement that checks whether a particular version of an Advanced UI or Suite/Advanced UI installation is already installed on a target system, enter the Suite GUID of that installation. ***Tip*** *• When you create an Advanced UI or Suite/Advanced UI project, InstallShield automatically creates a Suite Installed exit condition for the installation, and uses an asterisk (*) in the condition's Suite GUID and Version settings as placeholders for the project's own Suite GUID and version. InstallShield adds this built-in condition to prevent the Advanced UI or Suite/Advanced UI installation from installing over a future newer version of the installation. In general, this built-in condition should not be overridden, except in advanced scenarios in which changing the default behavior is appropriate.* *To learn more, see Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed.*

Table 9-20 • Suite Installed Condition Settings (cont.)

Setting	Description
Version	To create a conditional statement that checks whether a particular version of an Advanced UI or Suite/Advanced UI installation is already installed on a target system, enter the version number. Then use the Comparison setting to specify whether you want the product version to be less than, greater than, or equal to the value that you are specifying. The typical format for the version number is *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the aaa and bbb portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535. Note that although you can include the fourth field (*ddddd*) when you specify the product version, the Advanced UI or Suite/Advanced UI installation does not use this part of the product version to distinguish between different product versions.  ***Tip*** • *When you create an Advanced UI or Suite/Advanced UI project, InstallShield automatically creates a Suite Installed exit condition for the installation, and uses an asterisk (*) in the condition's Suite GUID and Version settings as placeholders for the project's own Suite GUID and version. InstallShield adds this built-in condition to prevent the Advanced UI or Suite/Advanced UI installation from installing over a future newer version of the installation. In general, this built-in condition should not be overridden, except in advanced scenarios in which changing the default behavior is appropriate.* *To learn more, see Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed.*

Table 9-20 • Suite Installed Condition Settings (cont.)

Setting	Description
Compare	Select the appropriate option to indicate how you want to compare the version number on the target system with the one that you enter. For example, if you want the condition to be true if the version on the target system is less than or equal to 4.0.0, enter 4.0.0 in the Product Version setting, and select Less Than or Equal To in the Comparison setting. ***Tip*** *• When you create an Advanced UI or Suite/Advanced UI project, InstallShield automatically creates a Suite Installed exit condition for the installation, and uses an asterisk (*) in the condition's Suite GUID and Version settings as placeholders for the project's own Suite GUID and version. InstallShield adds this built-in condition to prevent the Advanced UI or Suite/Advanced UI installation from installing over a future newer version of the installation. In general, this built-in condition should not be overridden, except in advanced scenarios in which changing the default behavior is appropriate.* *To learn more, see Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed.*

UWP App Package Condition Settings

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create a UWP App Package condition.

The UWP App Package setting shows the condition that is configured for checking target systems for the presence of a particular UWP app. To define this type of condition, click the ellipsis button (...) in the UWP App Package setting to browse to the package and have InstallShield enter the appropriate information in the subsettings for you. As an alternative, you can manually configure the subsettings under the UWP App Package setting as needed.

Note *• Support for browsing to the UWP app package (.appx) by using the ellipsis button in the UWP App Package setting requires Windows Vista or later or Windows Server 2008 or later on the machine that has InstallShield or the Standalone Build.*

The following subsettings are available for this condition.

Table 9-21 • UWP App Package Condition Settings

Setting	Description
Name	To create a conditional statement that checks for the presence of a particular UWP app, enter the name that is used to identify the UWP app to the operating system. To browse to the appropriate UWP app package (.appx) and have InstallShield enter the value in this setting for you, click the ellipsis button (...) in the UWP App Package setting.
Publisher	This setting specifies the publisher that is specified in the subject field of the certificate that was used to sign the package. To browse to the appropriate UWP app package (.appx) and have InstallShield enter the value in this setting for you, click the ellipsis button (...) in the UWP App Package setting.
Version	This setting specifies the version number of the package. To browse to the appropriate UWP app package (.appx) and have InstallShield enter the value in this setting for you, click the ellipsis button (...) in the UWP App Package setting.
Comparison	If you specify a version number in the Version setting, select the appropriate option to indicate how you want to compare the version number on the target system with the one that you enter. For example, if you want the condition to be true if the version on the target system is less than or equal to 4.0.0, enter 4.0.0 in the Version setting, and select Less Than or Equal To in the Comparison setting.
Processor Architecture	This setting identifies the architecture (x86, x64, or neutral) that must be present on target systems for this package. To browse to the appropriate UWP app package (.appx) and have InstallShield enter the value in this setting for you, click the ellipsis button (...) in the UWP App Package setting.

UWP Type Present Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

***Important** • This condition always evaluates as false on operating systems before Windows 10. Use of the Type Name subsetting `Windows.ApplicationModel.FullTrustProcessLauncher` requires Windows 10 Anniversary Update or newer.*

When you are building a conditional statement for an exit or eligibility condition in an Advanced UI or Suite/Advanced UI project, you can create a UWP Type Present condition.

The UWP Type Present setting shows the condition that is configured for checking target systems for the presence of UWP functionality. For example, to create a conditional statement that checks for the presence of the Desktop Bridge, check for the type `Windows.ApplicationModel.FullTrustProcessLauncher`. This can be used to conditionally block installation, or to choose between installing .msi and UWP app package (.appx) packages..

Table 9-22 • UWP Type Present Condition Setting

Setting	Description
Type Name	To create a conditional statement that checks for the presence of a particular UWP type, choose `Windows.ApplicationModel.FullTrustProcessLauncher` from the drop-down list or enter a name that is used to identify a UWP type to the operating system.

Package Operation Condition Settings

***Project** • This information applies to Suite/Advanced UI projects.*

***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an action condition in a Suite/Advanced UI project, you can create a Package Operation condition.

The Package Operation setting shows the condition that is configured for checking the target state of a particular package in a Suite/Advanced UI installation. This type of condition is available only for actions that are associated with an event in the Events view or Packages view. If you associate the action with an event in the Events view, the earliest event for which you can schedule the action is OnStaging.

To define this type of condition, configure the subsettings under the Package Operation setting as needed.

The following subsettings are available for this condition.

Table 9-23 • Package Operation Condition Settings

Setting	Description
Package	To create a conditional statement that checks the target operation of a particular package in the Suite/Advanced UI installation, select the name of the package in this setting.
Operation	Select the type of operation that you want to check for the package operation condition. Available options are: • **No Action**—The package is not set to be installed, removed, modified, or repaired. • **Install**—The package is set to be installed. • **Remove**—The package is set to be removed. • **Modify**—The package is set to be modified. • **Repair**—The package is set to be repaired.

Feature Operation Condition Settings



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an action condition in a Suite/Advanced UI project, you can create a Feature Operation condition.

The Feature Operation setting shows the condition that is configured for checking the target state of a particular feature in a Suite/Advanced UI installation. This type of condition is available only for actions that are associated with an event in the Events view or Packages view. If you associate the action with an event in the Events view, the earliest event for which you can schedule the action is OnStaging; feature states are not available before the OnStaging event.

To define this type of condition, configure the subsettings under the Feature Operation setting as needed.

The following subsettings are available for this condition.

Table 9-24 • Feature Operation Condition Settings

Setting	Description
Feature	To create a conditional statement that checks the target operation of a particular feature in the Suite/Advanced UI installation, select the name of the feature in this setting.
Operation	Select the type of operation that you want to check for the feature operation condition. Available options are: • **No Action**—The feature is not set to be installed or removed. • **Install**—The feature is set to be installed. • **Remove**—The feature is set to be removed.

Extension Condition Settings



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can create an extension condition that calls a C/C++ DLL that you have created to implement your own custom condition. The custom condition could check target systems for factors that the built-in types of condition checks do not evaluate. For instructions on how to create the DLL and add it to your project, see Creating a Custom Condition for an Advanced UI or Suite/Advanced UI Installation.

When you add an extension condition DLL, InstallShield adds a row for the condition. The row consists of two fields. The left field of this row uses the following syntax to describe your condition:

```
ExtensionDLLName:Condition
```

where *ExtensionDLLName* is the name of the DLL file, but without the .dll part of the name.

The right field of this row is a read-only field that shows the parameter-and-value pairs that you have defined for the condition. This setting also contains an **Add Parameter** button, which lets you define parameters and their values, as well as a **Delete this condition** button, which lets you delete the extension condition.

To add a parameter and value, click the **Add Parameter** button in the right field of the extension condition row. The **Configure Extension Condition** dialog box opens, enabling you to enter the name of the parameter. InstallShield adds a new row for the parameter. Configure the value of the parameter in the field on the right.

Mode Condition Settings



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for a wizard interface condition in an Advanced UI or Suite/Advanced UI project, you can create a Mode condition. The Mode setting shows the condition that is configured for checking the mode in which the Advanced UI or Suite/Advanced UI installation is running. To define this type of condition, configure the subsetting under the Mode setting as needed.

The following subsetting is available for this condition.

Table 9-25 • Mode Condition Settings

Setting	Description
Value	To create a conditional statement that checks the mode in which the Advanced UI or Suite/Advanced UI installation is running, select the appropriate mode in this setting. Available options are: • **Installing**—The installation is running in first-time installation mode. • **Staging**—The installation is being staged; that is, it is being uncompressed and copied to a specified location. Note that none of the packages in the installation are run in this mode. • **Removing**—The product is being removed. • **Modifying**—The product is being modified. • **Repairing**—The product is being repaired. • **Maintenance (remove, modify, repair)**—The installation is running in maintenance mode (modify, remove, or repair).

Guidelines for Defining Conditions in an Advanced UI or Suite/Advanced UI Project



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Following are some guidelines on defining conditions in an Advanced UI or Suite/Advanced UI project.

Define an Eligibility Condition for Each .exe Package to Prevent Downgrades

InstallShield automatically handles an empty eligibility condition as a request to ensure that a later version of the package is not already present on target systems. You can augment this, for example with a Platform condition to prevent installation on unsupported platforms, without losing the automatic behavior.

For .exe packages, create an appropriate eligibility condition to ensure that the Advanced UI or Suite/Advanced UI installation does not downgrade your product by allowing an earlier version to install over a later version.

Note that if your product supports side-by-side versions of, for example, version 3 with version 4 on the same system, ensure that your conditions support this. If your product does not support these versions on the same system, ensure that your conditions block this scenario.

Consider Defining an Eligibility Condition for Each Package to Check for Requirements and Dependencies

A package may fail to install or the product may not run as expected on target systems that do not meet its requirements (such as the minimum operating system version) or that do not have its dependencies (such as the .NET Framework, Microsoft SQL Server, or a third-party library). To prevent the Advanced UI or Suite/Advanced UI package from attempting to download (if applicable) and launch packages on a target system on which they will fail, define an eligibility condition that checks for requirements and dependencies. And whenever you have a package that contains one or more launch conditions, ensure that you add corresponding eligibility conditions to the package.

Because .msp packages require their target .msi package to be present, create an eligibility condition that checks target systems for the presence of the product that you want to update.

Review the Default Exit Condition for Relevance

By default, when you create a new Advanced UI or Suite/Advanced UI project, InstallShield automatically creates an exit condition that aborts the Advanced UI or Suite/Advanced UI installation if none of the primary packages are eligible for installation. The exit condition has an Eligible Package type of condition check, and it uses an asterisk (*) in the condition's package ID setting as a placeholder for each primary package's ID.

You can indirectly control this default exit condition by modifying the eligibility condition of each package.

The default exit message assumes that your eligibility conditions only prevent downgrades. If any of the primary packages in your project have an eligibility condition that checks for something else, you may need to edit the default message to properly reflect the reason for the Advanced UI or Suite/Advanced UI installation aborting.

For example, if your Advanced UI or Suite/Advanced UI project includes one 64-bit primary package, this package may have an eligibility condition that checks for an x64 architecture. In this case, you may need to edit the exit message and condition to account for the fact that the package is not eligible on a 32-bit target system. However, if your Advanced UI or Suite/Advanced UI package includes one 64-bit primary package and one 32-bit primary

package, and you configure the eligibility conditions for these packages so that the appropriate one is launched on target systems that have the corresponding architecture, the default exit message and condition may be suitable as is.

Define a Detection Condition for Each .exe Package

Every .exe package in an Advanced UI or Suite/Advanced UI project must have at least one defined detection condition. The detection condition should evaluate whether the package is already installed on target systems. At build time, InstallShield combines the detection conditions from all of the primary packages in an Advanced UI or Suite/Advanced UI project to form part of the mode conditions of the Advanced UI or Suite/Advanced UI installation.

The Advanced UI or Suite/Advanced UI **Setup.exe** file can determine whether .msi packages and InstallScript packages in the Advanced UI or Suite/Advanced UI installation have already been installed on target systems. It can also determine whether an .msp package in the Advanced UI or Suite/Advanced UI installation has already been applied to an earlier version of the product. Similarly, it can determine whether a UWP app package (.appx) in the Suite/Advanced UI installation have already been installed on target systems. Therefore, manually defining a detection condition for these three types of packages is not a requirement.

Be Very Explicit

Make sure that you are creating conditions for all of the possible results; that way, you have full control over the appropriate behavior. For example, if you are checking for a specific version of a file or for a particular registry value, make sure to also check for the existence of the file or registry entry. Sometimes a missing file is fine but a given version number of a file is bad; sometimes a missing file is bad.

Test, Test Some More, and Then Test Again

Test your conditions on various target systems and under different scenarios to make sure that the Advanced UI or Suite/Advanced UI installation behaves as expected. Use clean machines, machines that have earlier versions of your products, and machines that have the current version. Also consider building a future version of your product to test that your current installation does not downgrade future versions.

Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Advanced UI and Suite/Advanced UI projects include support for determining whether a particular version of an Advanced UI or Suite/Advanced UI installation is already installed on target systems. This type of condition check is called a *Suite Installed condition*, and it checks the target system's registry for the Advanced UI or Suite/Advanced UI installation's Uninstall key.

When an end user successfully installs a product through an Advanced UI or Suite/Advanced UI installation, the installation creates an Uninstall registry key in the following 32-bit location in the registry:

SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall*{Suite GUID}*

The Uninstall key contains uninstallation information for the installation. The name that is used for the registry key is the Suite GUID of the Advanced UI or Suite/Advanced UI installation. One of the registry values that this registry key contains is a DisplayVersion value; the data for this value is the version number of the Advanced UI or Suite/Advanced UI installation. Both the Suite GUID and the version number are defined in the General Information view of the project.

InstallShield includes two Suite Installed conditions in each Advanced UI or Suite/Advanced UI project by default. You can also add your own Suite Installed conditions to a project as needed.

Default Suite Installed Conditions in Advanced UI and Suite/Advanced UI Projects

When you create an Advanced UI or Suite/Advanced UI project, InstallShield creates two Suite Installed conditions in your project by default:

- **An exit condition**—This default exit condition is shown in the General Information view of the project. The Suite Installed exit condition prevents end users from being able to install the current version of the Advanced UI or Suite/Advanced UI installation over a future newer version of the same Advanced UI or Suite/Advanced UI installation.

  That is, the installation compares the version number that is stored in the DisplayVersion value of the Uninstall key on the target system with the value that is configured in the project. If the version in the target system's registry is greater than the version that is configured in the project, the installation displays an error message. When the end user dismisses the message, the installation exists. This default Suite Installed exit condition prevents downgrades of the Advanced UI or Suite/Advanced UI installation.

- **An install mode condition**—The install mode condition is shown only in the project file (.issuite). This type of condition is not available for edit within InstallShield. The Suite Installed mode condition causes the Advanced UI or Suite/Advanced UI installation to run in first-time installation mode if end users install a new version of the Advanced UI or Suite/Advanced UI installation over an older version of the same Advanced UI or Suite/Advanced UI installation. Furthermore, the Suite Installed mode condition may prevent the installation from running in first-time installation mode if the same version of the Advanced UI or Suite/Advanced UI installation is already installed.

  The default install mode condition compares the DisplayVersion value of the Uninstall key on the target system with the value that is configured in the project. If these values are not the same, the Suite Installed part of the install mode condition evaluates as true.

  Note that an install mode condition also evaluates the detection conditions of all of the primary packages in the installation. Thus, even though the Suite Installed part of the install mode condition may evaluate as false (meaning that it appears that the target system has the same version that is configured in the project), the Advanced UI or Suite/Advanced UI installation may run in first-time installation mode. This may occur if none of the detection conditions for the primary packages in the installation evaluate as true.

In general, these built-in Suite Installed conditions should not be overridden, except in advanced scenarios in which changing the default behavior is appropriate.

Note that InstallShield uses an asterisk (*) in the default Suite Installed condition's Suite GUID and Version settings as placeholders for the project's own Suite GUID and version. At build time, InstallShield replaces the asterisks with the appropriate values in the resulting build.

Using the Suite Installed Condition Check in Your Own Conditions

If you want your Advanced UI or Suite/Advanced UI installation to check target systems for the presence of a particular version of the installation, you can add your own Suite Installed condition check for an exit condition, a detection condition, or other any other condition.

Using Locale Conditions in Advanced UI and Suite/Advanced UI Projects



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The locale type of condition in an Advanced UI or Suite/Advanced UI project has several subsettings. The following table describes the formats that you can use for specifying the languages in each of these subsettings.

Table 9-26 • Subsettings Under the Locale Condition Setting

Setting	Format for Values in This Setting	Notes
User Interface	Enter locale identifiers (LCIDs) in any of the following formats: • Decimal LCID; for example: **1033** • Hexadecimal LCID preceded with 0x; for example: **0x409** • Locale name, which is checked only on Windows Vista and later systems, and ignored on earlier systems; for example: **en-us**, **en**, **zh-CN** You can optionally precede an LCID or name with a tilde (~) to check for a language-only match, ignoring the country or region. For example, an entry of **~1033**—for English (United States)—results in a match on a target system that uses Australian English, UK English, U.S. English, or other regional variants of English.	At run time, the installation uses the following functions to check the user interface on target systems: • GetUserDefaultLCID • GetSystemDefaultLCID • GetUserDefaultLocaleName (only on Windows Vista and later systems) • GetSystemDefaultLocaleName (only on Windows Vista and later systems) For a list of LCIDs, see Locale IDs Assigned by Microsoft on MSDN.
Operating System	Enter LCIDs in either of the following formats: • Decimal LCID; for example: **1033** • Hexadecimal LCID preceded with 0x; for example: **0x409** You can optionally precede an LCID with a tilde to check for a language-only match, ignoring the country or region. For example, an entry of **~1033**—for English (United States)—results in a match on a target system that uses Australian English, UK English, U.S. English, or other regional variants of English.	One example of when this condition is useful is for .NET Framework 1.1 service packs for Windows Server 2003; these service packs target specific operating system languages. At run time, the installation uses the function GetSystemDefaultUILanguage to check the language of the operating system on target systems. For a list of LCIDs, see Locale IDs Assigned by Microsoft on MSDN.

Table 9-26 • Subsettings Under the Locale Condition Setting (cont.)

Setting	Format for Values in This Setting	Notes
ANSI Code Page	Code page identifiers; for example: **932**	At run time, the installation uses the GetACP function to check the ANSI code page of the target system. Check the Knowledge Base for information about this error, or request technical support.For a list of code page identifiers, see Code Page Identifiers on MSDN.

If you want to check the user interface, operating system, or ANSI code page on target systems for any one of two or more languages, you can specify multiple values and separate them with a comma (,). Any spaces that are included in the setting are ignored. For example, to check for either a U.S. English or Japanese user interface, you could enter the following value in the User Interface setting:

0x409, 0x411

In this scenario, the User Interface condition is met if the target system's UI is U.S. English or Japanese.

Note that if you specify values in two or more locale settings, each of those locale settings must have a match in order for the condition to be met. For example, if you specify languages in the User Interface and ANSI Code Page settings, the condition evaluates as true whenever one of the User Interface languages are a match and also one of the ANSI Code Page languages are a match.

Creating a Custom Condition for an Advanced UI or Suite/Advanced UI Installation



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

When you are building a conditional statement for an exit, detection, eligibility, or feature condition in an Advanced UI or Suite/Advanced UI project, or for an action condition in a Suite/Advanced UI project, you can select from a number of different built-in types of checks that you want to be evaluated on target systems. Examples are operating system version, presence of a file or registry entry, and presence of an already installed product.

The built-in types of condition checks may not cover all of the scenarios that you want your Advanced UI or Suite/Advanced UI installation to evaluate on target systems. Therefore, InstallShield lets you create your own type of custom condition called an *extension condition*. An extension condition calls a C/C++ DLL that you have created.

For example, if you want to check for the existence of an IIS Web site, you can create a DLL that checks target systems and then include this DLL in your Advanced UI or Suite/Advanced UI project.

For guidance on how to create the extension condition DLL, see Creating an Extension Condition DLL for an Advanced UI or Suite/Advanced UI Project.

For instructions on how to incorporate the extension condition DLL in your project, see Adding an Extension Condition DLL to an Advanced UI or Suite/Advanced UI Project.

Creating an Extension Condition DLL for an Advanced UI or Suite/Advanced UI Project



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*



***Note** • Flexera Software does not provide technical support for Windows programming or DLL debugging. You are responsible for correctly writing any DLL functions. Prototype your custom DLL functions as explained below. Any variation in return type can cause the extension condition DLL to fail.*

The first step in creating an extension condition is to create the DLL. You can create an extension condition DLL with any recent version of Visual C++ or any tool or language that supports COM and DLL exports.

The Advanced UI and Suite/Advanced UI engine requires two entry points per condition—one to validate the condition and one to evaluate the condition. Both entry points must be based on the name of the extension condition. For example, for a condition called **MyCondition**, the DLL that tests the condition must export the following entry points:

- **MyCondition_Validate**
- **MyCondition_Evaluate**

These functions should be defined as follows:

```
HRESULT __stdcall MyCondition_Validate(IDispatch *pCondition);
HRESULT __stdcall MyCondition_Evaluate(IDispatch *pCondition, VARIANT_BOOL *pbResult);
```

To be able to call these functions from the Advanced UI or Suite/Advanced UI installation, you must include a definition file (.def) when you build the DLL to export the function properly. The following is the contents of a sample .def file. The name after LIBRARY should be the name that you are using for the DLL.

```
LIBRARY "MyConditionLibrary"
EXPORTS
    MyCondition_Validate
    MyCondition_Evaluate
```

The Advanced UI or Suite/Advanced UI installation calls the validation entry point while parsing the conditions that were defined in the project. This enables the extension condition to ensure that the condition parameters that were defined in the project are suitable for the condition to evaluate in a predictable manner. This entry point is called any time an extension condition of this type is found for an exit condition, a detection condition, or any other type of condition. Returning S_OK from this entry point indicates to the Advanced UI or Suite/Advanced UI installation that the condition is valid. Returning a failed HRESULT status causes the Advanced UI or Suite/Advanced UI installation to abort with an error for the condition that was last validated.

The Advanced UI or Suite/Advanced UI installation calls the evaluation entry point any time that an extension condition of this type is referenced by a condition that was configured in the project. The evaluation is allowed to perform any action that is necessary to evaluate the condition to a true or false result. The result is returned in the form of a VARIANT_BOOL in the pbEvalResult parameter. A result of VARIANT_TRUE is treated as a true result and a result of VARIANT_FALSE is treated as a false result. A success status (S_OK) should be returned from an evaluation function. A failure status aborts the evaluation of the current condition block being evaluated, but it does not cause the installation to abort.

Note that although the Advanced UI or Suite/Advanced UI installation allows any action to be performed, extension conditions run with the privileges of the Advanced UI or Suite/Advanced UI installation from which they are called. Some actions may require administrator privileges (such as reading IIS 7.x configuration data). In such cases, in order for the condition to access the required data successfully, the installation would need to be run with administrator privileges, or it would need to include an administrator manifest.

The IDispatch interface that is passed to the validation and evaluation entry points implements the ISuiteExtension interface. To obtain a pointer to an ISuiteExtension, call the QueryInterface method on the IDispatch interface. The ISuiteExtension interface allows the condition functions to access the attribute parameters that are defined for the extension condition in the Advanced UI or Suite/Advanced UI project. Note that each condition in the project receives a different interface pointer that is specific to each condition instance. Therefore, the interface pointer that is passed to the entry point function should always be used and should not be saved across calls to the extension DLL.

The interface is defined as follows:

```
interface ISuiteExtension : IDispatch {
  [propget, id(1), helpstring("Attribute")]
    HRESULT Attribute([in]BSTR bstrName, [out, retval]BSTR *bstrValue);

  [id(2), helpstring("method LogInfo")]
    HRESULT LogInfo([in]BSTR bstr);

  [propget, id(3), helpstring("Property")]
    HRESULT Property([in]BSTR bstrName, [out, retval]BSTR *bstrValue);

  [propput, id(3), helpstring("Property")]
    HRESULT Property([in]BSTR bstrName, [in]BSTR bstrValue);

  [id(4), helpstring("FormatProperty")]
    HRESULT FormatProperty([in]BSTR bstrValue, [out, retval]BSTR *bstrReturnValue);

  [id(5), helpstring("ResolveString")]
    HRESULT ResolveString([in]BSTR bstrStringId, [out, retval]BSTR *bstrReturnValue);
};
```

Advanced UI and Suite/Advanced UI projects use the function prototype to pass the ISuiteExtension COM interface pointer to an extension condition DLL, which enables you to access the following functions:

- **get_Attribute**

```
HRESULT get_Attribute([in]BSTR bstrName, [out, retval]BSTR *bstrValue);
```

This method retrieves the value of an attribute for the current condition instance. The attribute name is passed in bstrName, and its corresponding value is returned in bstrValue.

- **LogInfo**

  ```
  HRESULT LogInfo([in]BSTR bstr);
  ```

  This method writes any information that is needed to the Advanced UI or Suite/Advanced UI debug log, making it available for debugging or other informational purposes. The bstr parameter contains the string to be written to the log.

- **get_Property**

  ```
  HRESULT get_Property([in]BSTR bstrName, [out, retval]BSTR *bstrValue);
  ```

  This method retrieves the value of any property that is currently defined in the running Advanced UI or Suite/Advanced UI installation. Properties that are not defined return an empty value. The bstrName parameter specifies the name of the property whose value is being obtained. The value for the property is returned in the bstrValue parameter.

- **put_Property**

  ```
  HRESULT put_Property([in]BSTR bstrName, [in]BSTR bstrValue);
  ```

  This method sets the value of a new property or changes the value of an existing property in the currently running Advanced UI or Suite/Advanced UI installation. Passing an empty value effectively deletes the property. The bstrName parameter specifies the name of the property to set. The bstrValue parameter specifies the value to set for the specified property.

- **FormatProperty**

  ```
  HRESULT FormatProperty([in]BSTR bstrValue, [out, retval]BSTR *bstrReturnValue);
  ```

  This method resolves embedded formatted expressions in the string that is provided in the bstrValue parameter. The formatted value is returned in the bstrReturnValue parameter. To learn about the syntax that is available for formatted expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

- **ResolveString**

  ```
  HRESULT ResolveString([in]BSTR bstrStringId, [out, retval]BSTR *bstrReturnValue);
  ```

  This method resolves an Advanced UI or Suite/Advanced UI string identifier into the corresponding string value for the currently running Advanced UI or Suite/Advanced UI installation in the currently selected UI language. The bstrStringId parameter specifies the string identifier to resolve, and the resolved string is returned in bstrReturnValue. If no such string identifier exists, the returned string is empty.

To access the ISuiteExtension interface from your DLL, you can use `#import` to incorporate the type library information from the **SetupSuite.exe** file that is installed with InstallShield. The path is:

*`InstallShield Program Files Folder`***\Redist\Language Independent\i386\SetupSuite.exe"**

For example, to import the type library in a VC++ project (such as in **stdafx.h**), use the following statement:

```
#import "C:\Program Files\InstallShield\2016\Redist\Language Independent\i386\SetupSuite.exe"
no_namespace raw_interfaces_only named_guids
```

Note that if InstallShield is installed to a different location, adjust the path in the #import statement accordingly.

Once you have created your extension condition DLL, you can add it to your project. To learn how, see Adding an Extension Condition DLL to an Advanced UI or Suite/Advanced UI Project.

Adding an Extension Condition DLL to an Advanced UI or Suite/Advanced UI Project



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Once you have created an extension condition DLL, you can add it to your project.



Task ***To add an extension condition DLL to an Advanced UI or Suite/Advanced UI project:***

1. Find the setting for the detection condition, eligibility condition, or other condition in your project that you want to use your extension condition DLL.

   To learn more about setting up these conditions, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.

2. If necessary, add an **Any**, **All**, or **None** operator to the condition group.

3. In the **Any**, **All**, or **None** setting, click the **New Condition** button and then click **Browse for an Extension Condition DLL**. The **Open** dialog box opens.

4. Select the extension condition DLL that you created, and then click **Open**. The **Configure Extension Condition** dialog box opens.

   Note that the DLL must be written according to the instructions in Creating an Extension Condition DLL for an Advanced UI or Suite/Advanced UI Project.

5. Click the condition that you want to use. InstallShield adds the DLL to your project as a support file. InstallShield also adds a new row for the condition. The row consists of two fields. The left field of this row uses the following syntax to describe your condition:

   `ExtensionDLLName:Condition`

   where *`ExtensionDLLName`* is the name of the DLL file, but without the .dll part of the name.

   The right field of this row is a read-only field that shows the parameter-and-value pairs that you have defined for the condition. This setting also contains an **Add Parameter** button, which lets you define parameters and their values, as well as a **Delete this condition** button, which lets you delete the extension condition.

6. To add a parameter and value, click the **Add Parameter** button in the right field of the extension condition row. The **Configure Extension Condition** dialog box opens.

7. Enter the name of the parameter, and then click **OK**. InstallShield adds a new row for the parameter. Configure the value of the parameter in the field on the right.

Searching for Installed Data



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield provides support for the Windows Installer capability of searching for installed data on a target system at run time. Depending on the results of the search, the installation can store a value in a property and even use the property in an installation condition. Examples of system searches that your installation can perform include:

- Search for a file or folder on the target system at run time and store the full path to the file or folder, if found, in a property. This includes support for XML files.
- Read data from the registry or from an .ini file located in the target system's WindowsFolder directory and store the data in a property. Then use the property in an installation condition.

InstallShield includes several predefined system searches that you can add to your project. In addition, system searches that have been defined in one project and published to a repository can be reused in other projects. Use the System Search view to add a predefined system search—whether it is a search that is included with InstallShield or one that is stored in a repository—to your project. You can also use the System Search view to customize any predefined searches or define your own system searches for your installation.

Adding a Predefined System Search



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Use the System Search view to add a predefined system search—whether it is a search that is included with InstallShield or one that is stored in a repository—to your project.

Task ***To add a predefined system search:***

1. Open the **System Search** view.
2. Right-click the grid and then click **Add predefined search**. The Add Predefined Search Dialog Box opens.
3. If you want system searches that have been published to a repository to be displayed, select the **Show Searches in Repository** check box.
4. Select the value for the system search that you would like to add.
5. Click **OK**.

InstallShield adds the search to the grid in the System Search view. If you need to make changes to the predefined search after you have added it to your project, see Modifying a System Search to learn how.

Adding a System Search

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Your installation can include system searches for different types of installed data on a target system. To add to your project one of the predefined system searches available in InstallShield or a system search that is stored in a repository, see Adding a Predefined System Search. To define your own custom system search, follow the steps below.

Task ***To add a system search:***

1. Open the **System Search** view.
2. Right-click the grid and then click **Add**. The **System Search Wizard** opens.
3. Complete the panels in the System Search Wizard.

InstallShield adds the search to the grid in the **System Search** view.

Modifying a System Search

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



Task ***To modify a system search:***

1. Open the **System Search** view.
2. Right-click the grid entry for the system search that you want to modify and then click **Modify**. The **System Search Wizard** opens.
3. Complete the panels in the System Search Wizard.

Special Considerations for Searching the Registry at Run Time



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The System Search Wizard and the System Search view enable you to configure the following types of registry-related system searches:

- File path, as specified by a registry entry
- Folder path, as specified by a registry entry
- Registry entry

Note the following details about these types of system searches.

Searching for the Presence of a Registry Key

Windows Installer does not have built-in support for searching target systems for the presence or absence of a specific registry key. However, you can create a system search that looks for the default value of a specific registry key. If the default value is present, Windows Installer writes it to the system search's associated property. If the system search fails to find the default value, Windows Installer leaves the property undefined.

Storing Registry Entry Types in a System Search's Property

At run time, Windows Installer sets the value of the property that you specified for your registry-related system search to the location that is found on the target system. In addition, Windows Installer adds a prefix to the property value to indicate the type of registry entry (such as REG_DWORD or REG_BINARY). If you then use the property to configure the registry, or if you create a condition that reads this property, you must understand the syntax that Windows Installer uses to set the property.

The following table lists each of the types of registry entries, along with the associated prefix that Windows Installer uses when setting the system search property.

Table 9-1 • Prefixes or Other Notation that Windows Installer Uses for a System Search Property's Value

Prefix/Notation	Meaning
<No prefix>	The property value contains a string value (REG_SZ).
# **#+** **#-**	The property value contains an integer value (REG_DWORD).
#x	The property value contains a hexadecimal value (REG_BINARY). Windows Installer converts each hexadecimal digit as an ASCII character for this property value.
#%	The property value contains an expandable string value (REG_EXPAND_SZ).
<null>	The property value contains a list of strings (REG_MULTI_SZ). This value starts and ends with null.

Publishing a System Search to a Repository

If your project contains a system search that you would like to reuse in other projects or share with other users, you can publish it to a repository.



Task ***To publish a system search to a repository:***

1. Open the **System Search** view.
2. In the grid, right-click the system search and then click **Publish Wizard**. The **Publish Wizard** opens.
3. Complete the panels in the Publish Wizard.

After you have imported a system search from a repository into a project, there is no link between the current system search and the existing repository system search. If you make a change to the system search and then republish it to the repository, it does not affect the system search in the project to which it was imported. However, you can reimport the system search from the repository into your project.

Removing a System Search



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



Task ***To remove a system search:***

1. Open the **System Search** view.
2. Right-click the grid entry for the system search that you want to remove and then click **Delete**.

The deleted system search will no longer be included in your installation.

Searching for XML Data



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Application and configuration settings are sometimes stored in XML files. Use the System Search view to define a search for data in an XML file that may exist on the target system at run time. You can search for a specific attribute, a specific element, or a specific element content string. The results are stored in a property; you can use that property in an installation condition, as the destination for a component in your installation, or in other areas of your installation.

Example

This section presents an example that shows how to search for XML data on target systems at run time. For this example, the product called **MyProduct** has an executable file called **MyProduct.exe**. The path for this .exe file is written to a file called **Configuration.xml** at run time.



Tip *• For information on how to use properties to change XML files at run time, see Using Windows Installer Properties to Dynamically Modify XML Files.*

Following is sample code from the **Configuration.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <requiredRuntime version="1.0.0" safemode="true" />
  </startup>
  <appSettings>
    <add value="MyProduct" path="C:\Program Files\MyCompany\MyProduct\MyProduct.exe"/>
  </appSettings>
</configuration>
```

The value of the **path** attribute in the above XML code depends on where end users install **MyProduct**.

The main objective of this example is to find the value of the **path** attribute and store that value in a property.



Task ***To configure the system search for the XML data:***

1. In the View List under **Behavior and Logic**, click **System Search**.
2. Right-click the grid and then click **Add**. The System Search Wizard opens.
3. Complete each of the wizard panels as follows:
   a. On the **What do you want to find?** panel: In the list, select **XML file value**.
   b. On the **Specify the file details** panel:
      - In the **File Name** box, type the following:

        Configuration.xml
      - In the **Look In** area, select **A full path**, and type the following path:

        [ProgramFilesFolder]MyCompany\MyProduct
   c. On the **Specify the data** panel:
      - In the **XPath to XML Element** box, type the following XPath expression:

        /configuration/appSettings/add[@value='MyProduct']
      - In the **Look For** area, select **Value of Attribute in this Element**, and type the following in the **Attribute Name** box:

        path
   d. On the **What do you want to do with the value?** panel:
      - In the **Store the value in this property** box, type the following:

        PATH_IN_XML_FILE
      - In the **Additional Options** area, select **Just store the value in the property**.

You can use the value of the search property in another area of your project. For example, to display the value of the property in a dialog in your installation, add a text control to the dialog, and set the Text value of the control to something like this:

PATH_IN_XML_FILE = [PATH_IN_XML_FILE]

If the value of the path attribute in the XML file is **C:\Program Files\MyCompany\MyProduct\MyProduct.exe**, the dialog displays the following text:

PATH_IN_XML_FILE = C:\Program Files\MyCompany\MyProduct\MyProduct.exe

Editing Installation and Project Tables

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

Important *• Editing the installation and project tables directly, rather than using the dedicated views in InstallShield, is for advanced users who are very familiar with the Windows Installer database format.*

The Windows Installer databases that are built with InstallShield consist of tables. InstallShield project files (.ism files) use the Windows Installer table format. In addition to providing a graphical user interface for working with the content of the Windows Installer and .ism tables, InstallShield also lets you manipulate the tables directly through the Direct Editor view.

The Direct Editor supports different types of tables:

- Standard tables, which are defined by Windows Installer
- InstallShield tables, which InstallShield creates to add functionality to your installation
- Custom tables, which you create and which you can read from or write to with a custom action

The Direct Editor can run in two different modes:

- **Project edit mode**—This mode lets you edit tables in the project file (.ism). The changes that are made in the Direct Editor are incorporated into the Windows Installer package that InstallShield creates at build time.

  When you are working in any of the following Windows Installer–based project types, you are working in project edit mode: Basic MSI, DIM, InstallScript MSI, Merge Module, and QuickPatch.

- **Direct edit mode**—This mode lets you edit tables in the Windows Installer database (.msi, .msm, or .mst file). When you save the changes that you have made in the Direct Editor, InstallShield updates the Windows Installer database.

  When you are working in any of the following Windows Installer–based project types, you are working in direct edit mode: MSI Database, MSM Database, and Transform.

Editing Project Tables

Project *• The Direct Editor in the following project types lets you view and edit the tables in your project file (.ism):*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *QuickPatch*

The Direct Editor in InstallScript and InstallScript Object projects lets you see all of the tables in your .ism file; however, it is recommended that you use the other views in InstallShield to modify these types of projects.

InstallShield project files (.ism files) use the Windows Installer table format. The Direct Editor is a view in InstallShield that lets you review all of the tables in your project file (.ism). This view also offers functionality for advanced users who are very familiar with the Windows Installer database format. For example, you can use the Direct Editor to perform the following tasks:

- View all of the tables in the project file (.ism).
- Add and remove records from tables.
- Cut, copy, and paste records or fields.
- Edit individual fields in the tables.
- Add custom tables.
- Filter the table records that are shown to hide ones that do not contain a specific string.
- Search one table or all tables for a specific string and replace it if necessary.
- Resize and reorder the columns in a table.

Adding New Tables Through the Direct Editor



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

InstallShield lets you add tables to your project file (.ism) or Windows Installer database (.msi, .msm, or .mst).



Task ***To add a new table through the Direct Editor:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. Right-click the **Tables** explorer and then click **Add Table**. The **Create Table Wizard** opens.

3. Complete all of the panels in the wizard.



Windows Logo • *The Windows logo program requires that you avoid naming a custom table with the MSI (in uppercase, lowercase, or mixed-case letters). The MSI prefix is reserved for future use in new standard properties and tables.*

Adding Records to a Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

The Direct Editor view lets you add records (or rows) to a table in your project or database.



Task ***To add a record (row) to a table:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. Select the table to which you want to add a record.
3. Click the **New Record** button. The **Add Record to Table** dialog box opens.
4. Complete each of the fields as needed.
5. Click **OK**.

Finding and Replacing in the Direct Editor



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

- *QuickPatch*
- *Transform*

The Direct Editor includes support for standard find-and-replace functionality. This functionality may be useful if you want to search for specific data or strings in your project or database and replace them with revised data or strings.

Finding Data in the Direct Editor

Task ***To find data in the Direct Editor:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, select one of the tables that you want to search.
3. Click the **Find String** button. The **Find** dialog box opens.
4. In the **Find what** box, type the string that you want to find.
5. Select any other options that you want.

   To search only the selected table, select the **Search in current table only** check box. To search all tables in the Direct Editor, clear this check box.
6. Click **Find Next**.

InstallShield finds the first instance of the string that you specified.

Finding and Replacing Data in the Direct Editor

Task ***To find and replace data in the Direct Editor:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, select the table in which you want to find and replace data.
3. Click the **Find and Replace** button. The **Replace** dialog box opens.
4. In the **Find what** box, type the string that you want to replace.
5. In the **Replace with** box, type the new string.
6. Select any other options that you want.
7. Click **Find Next**, **Replace**, or **Replace All**.

InstallShield replaces strings as needed.

Editing a Table Record

Project • *This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

The Direct Editor view lets you edit a record—or row—in a table of your project or database.

Task ***To edit a record in a table through the Direct Editor:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, select the table that contains the record that you want to edit, and then select the row.
3. Click the **Edit Selected Record** button. InstallShield displays the **Edit Record in Table** dialog box. This dialog box lets you edit any of the data in the selected row.
4. Edit the record as needed.
5. Click **OK**.

Note *• The Direct Editor performs field-level validation to help you avoid using an improper data type.*

To maintain referential integrity when editing tables in the Direct Editor, select the ***Maintain referential integrity*** *check box on the Preferences tab of the Options dialog box. You can access this dialog box by clicking Options on the Tools menu. The check box is selected by default.*

Editing Fields in a Table

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

The Direct Editor view contains spreadsheetlike tables that let you modify strings and other data in your project or database.

Task ***To edit fields in a table through the Direct Editor:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. Select the table that contains the field that you want to edit.
3. Do any of the following:
   - To overwrite all of the text in a table cell, click the table cell and then type the new value.
   - To place the cursor at a particular place within a table cell, double-click that place. Then type your change.
   - If a field takes a foreign key into another table, you can select the appropriate foreign key from the list in the cell.

Tip *• The parenthetical note in each column heading indicates the type and size of data that you can enter in the column. For example, S255 indicates a string with up to 255 characters; I2 indicates a 2-byte integer.*

Note *• The Direct Editor performs field-level validation to help you avoid using an improper data type.*

To maintain referential integrity when editing tables in the Direct Editor, select the ***Maintain referential integrity*** *check box on the Preferences tab of the Options dialog box. You can access this dialog box by clicking Options on the Tools menu. The check box is selected by default.*

Exporting Tables from the Direct Editor

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

The Direct Editor lets you export one or more tables as .idt files. You can then import the .idt files into another project or database through the Direct Editor.

Task ***To export one or more tables from the Direct Editor:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, right-click one of the tables that you want to export, and then click **Export Table(s)**. The **Export Table(s)** dialog box opens.
3. In the **Output Directory** box, specify the directory where you want InstallShield to save the .idt files, or click the **Browse** button to navigate to the location.
4. In the **Tables** box, select the check boxes of the tables that you want to export.
5. Click **OK**.

InstallShield creates a separate .idt file for each table that you selected to export.

Importing Tables with the Direct Editor

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

You can use the Direct Editor to import a table (.idt file) into your InstallShield project file (.ism) or Windows Installer database (.msi, .msm, or .mst).

Task ***To import a table (.idt file) using the Direct Editor:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. Right-click the **Tables** explorer and click **Import Table**.
3. Select the table that you want to import.
4. Click **Open**.

InstallShield imports the table into your project or database.

Deleting Records from a Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

You can use the Direct Editor to delete a record in your InstallShield project file (.ism) or Windows Installer database (.msi, .msm, or .mst).



Task ***To delete a record (row) from a table:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, select the table that contains the record that you want to delete.
3. In the table grid, select the record that you want to delete.
4. Click the **Delete Selected Records** button or press DELETE.

Removing Tables Through the Direct Editor



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

You can use the Direct Editor to delete a custom table in your InstallShield project file (.ism) or Windows Installer database (.msi, .msm, or .mst).

Task ***To remove a custom table:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. In the **Tables** explorer, right-click the custom table that you want to delete, and then click **Drop Table**.

Note • *You can remove (drop) only user-defined tables. InstallShield tables and standard tables cannot be removed.*

In-Place Windows Installer Database and Project File Differencing

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

InstallShield provides .msi file differencing functionality, which allows you to have two .msi packages open in the Direct Editor for comparison purposes. When you are editing a transform (.mst file), the differences between the base package and the applied transform are highlighted in the Direct Editor.

The base package is the current file that you have open in the InstallShield interface. The compared file is any package you have on your system that you want to compare with the base package. Therefore, if you see a red X icon in front of a table in the Direct Editor after you initiate a difference, then that suggests that the table exists in the base file and not the compared file.

In-place differencing is not just limited to .msi packages and transforms. You can also compare other files such as .msm, .ism, and .pcp files.

In the Direct Editor, you can undo or accept the difference. Icons in the first column of the Direct Editor tables indicate whether the row is a deletion or an addition, or if the row contains a change. The comparison file also displays added or removed columns in a table.

Table 9-1 • Icons in the Direct Editor Tables

Icon	Description
	Row has been changed.
	Row has been added.

Table 9-1 • Icons in the Direct Editor Tables (cont.)

Icon	Description
	Row has been deleted.

To see the specific change that has been made, move your mouse over a column to see a tooltip that describes the difference.



Task ***To compare two projects or databases:***

1. Open the base project file (.ism) or database (.msi, .msm, .mst, or .pcp).
2. On the **Tools** menu, point to **Difference**, and then click **Compare to File**. The **Select Compare File** dialog box opens.
3. In the **Files of type** box, select the type of file that you want to compare with your base file, and then browse to select it.
4. Click **Open**.

InstallShield indicates any differences between the two files in the Direct Editor.



Task ***To end the compare session:***

On the **Tools** menu, point to **Difference**, and then click **End Compare**.

InstallShield removes all of the in-place differencing from the Direct Editor.

Entering Binary Data in a Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

When you are working in any of the following Windows Installer–based project types, you are working in project edit mode: Basic MSI, DIM, InstallScript MSI, Merge Module, and QuickPatch. In project edit mode, tables such as the **Binary**, **Icon**, and **Patch** tables do not store binary data; they store links to build-source paths. Therefore, if you use the Direct Editor to edit these tables, ensure that you include the fully qualified path and file name. Otherwise, InstallShield cannot include the file in the release that it builds.

When you are working in any of the following Windows Installer–based project types, you are working in direct edit mode: MSI Database, MSM Database, and Transform. In these project types, tables such as the **Binary**, **Icon**, and **Patch** tables store binary data. If you use the Direct Editor to edit these tables, InstallShield enables you to browse to select the file that should be streamed into the table.

Orphaned Directory Entries



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

Directory entries are orphaned when they are improperly or inappropriately modified—possibly in the Direct Editor. This happens when the path is broken in the **Directory** table.

For example, **INSTALLDIR** is built by combining the three records that make up the full path:

ProgramFilesFolder\ISYourCompanyDir\ISYourProductDir

This typically resolves to:

C:\Program Files\Your Company Name\Your Product Name

If any record of **ProgramFilesFolder\ISYourCompanyDir\ISYourProductDir** is modified to be invalid, **INSTALLDIR** or any other directory path that contains the invalid record will not be resolved correctly. In this example, if you rename the **Directory** record ISYourCompanyDir to MyCompany (in the Directory column), but do not update the Directory_Parent of the ISYourProductDir record to MyCompany, the **INSTALLDIR** path is broken.

Using InstallShield Tools

InstallShield includes various tools to help you troubleshoot complex issues with Windows Installer packages and InstallScript installations. To find out more, refer to this section of the documentation.

InstallShield MSI Tools



Edition • *The InstallShield MSI tools are included with the Premier and Professional editions of InstallShield. The Premier edition also includes a separate installation and extra licenses that let you install just the tools, without InstallShield, on separate machines. For specific terms, see the End-User License Agreement for the InstallShield MSI tools.*

InstallShield includes tools that help you troubleshoot complex issues with Windows Installer packages:

- InstallShield MSI Diff enables you to quickly compare two .msi, .msm, or .pcp files. It lets you apply one or more .msp and .mst files to an .msi file and see the changes in the resulting .msi database. You can also use this tool to compare two InstallShield project files (.ism or .ise) that are saved in binary format. This tool uses color coding to show additions, modifications, deletions, and schema differences. You can easily integrate InstallShield MSI Diff with most source code control systems.
- InstallShield MSI Query lets you test SQL statements using the Windows Installer version of SQL before you run them in your build script. You can quickly see if a SQL statement is formatted correctly and view the results that it generates.
- InstallShield MSI Sleuth is a diagnostic tool that lets you view the current installed state of a target system. InstallShield MSI Sleuth displays a list of all of the .msi packages that are installed. You can click any .msi package in the list to see the status of its features and components, its known source locations, as well as tables and binary streams within the database. This tool also helps you identify the installed product or products whose packages contain a specific component code.
- InstallShield MSI Grep searches a collection of .msi and .msm packages for specific text. You can narrow the search to show results for only certain tables or columns.

You can launch any of these tools from the InstallShield Tools subfolder on the Windows Start menu.



Note • *All of these tools require .NET Framework 2.0 to be installed.*

InstallShield MSI Diff

InstallShield MSI Diff enables you to quickly compare two .msi, .msm, or .pcp files. It lets you apply one or more .msp and .mst files to an .msi file and see the changes in the resulting .msi database. You can also use this tool to compare two InstallShield project files (.ism or .ise) that are saved in binary format. You can easily integrate it with most source code control systems.

This tool uses color coding to show additions, modifications, deletions, and schema differences. The color code in the lower-right corner of InstallShield MSI Diff shows the legend that identifies which colors indicate each type of difference.

You can launch InstallShield MSI Diff from the InstallShield Tools subfolder on the Windows Start menu. You can also launch this tool from within InstallShield: On the Tools menu, point to Difference, and then click InstallShield MSI Diff.



Tip *• To show or hide unchanged rows in the various database tables: On the View menu, click Unchanged Rows. This menu command has a check mark if the unchanged rows are displayed; it does not have a check mark if the unchanged rows are hidden.*

To show or hide unchanged database tables: On the View menu, click Unchanged Tables. This menu command has a check mark if the unchanged tables are displayed; it does not have a check mark if the unchanged tables are hidden.

Determining the Changes that a Transform (.mst File) Makes

You can use InstallShield MSI Diff to see how an .msi package changes when a transform (.mst file) is applied to it.



Task ***To determine the changes that a transform makes:***

1. Open InstallShield MSI Diff.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.
3. Select the .msi package, and then click **Open**. InstallShield MSI Diff opens the .msi database.
4. On the **File** menu, point to **Transforms**, and then click **Apply Transform**. The **Open** dialog box opens.
5. Select the .mst transform, and then click **Open**. InstallShield MSI Diff applies the transform to the .msi database that is open.
6. Examine the changes.

If you want to apply a series of transforms to an .msi package, you can repeat steps 4 and 5 for each transform.



Tip *• Note that InstallShield MSI Diff also lets you apply a series of multiple transforms and patches to an .msi package, and see the changes in the resulting .msi database.*

To learn how to apply a patch to an .msi package, see Determining the Run-Time Effects of a Patch.

Determining Whether a Patch Is Valid for an .msi Package

You can use InstallShield MSI Diff to determine whether a patch is valid for a specific .msi package. InstallShield MSI Diff analyzes the internal transforms that are contained within the patch.



Task ***To determine whether a patch is valid for an .msi package:***

1. Open InstallShield MSI Diff.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.
3. Select the .msi package, and then click **Open**. InstallShield MSI Diff opens the .msi database.

4. On the **File** menu, point to **Patches**, and then click **Apply Patch**. The **Open** dialog box opens.
5. Select the patch (.msp file), and then click **Open**.

If the patch applies to the selected .msi database, InstallShield MSI Diff applies the patch to the database that is open. If the patch is not applicable, InstallShield MSI Diff displays a message box to inform you that the patch is not valid for the selected database.

Determining the Run-Time Effects of a Patch

You can use InstallShield MSI Diff to determine what Windows Installer will do with a given patch file and how that will affect the cached .msi package on a target system.



Task ***To determine the run-time effects of a patch:***

1. Open InstallShield MSI Diff.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.
3. Select the .msi package, and then click **Open**. InstallShield MSI Diff opens the .msi database.
4. On the **File** menu, point to **Patches**, and then click **Apply Patch**. The **Open** dialog box opens.
5. Select the patch (.msp file), and then click **Open**. InstallShield MSI Diff applies the patch to the database that is open.
6. Examine the changes.

If you want to apply a series of patches to an .msi package, you can repeat steps 4 and 5 for each patch.



Tip *• Note that InstallShield MSI Diff also lets you apply a series of multiple transforms and patches to an .msi package, and see the changes in the resulting .msi database.*

To learn how to apply a transform to an .msi package, see Determining the Changes that a Transform (.mst File) Makes.

Determining the Differences Between Two .msi Packages

You can use InstallShield MSI Diff to determine the differences between two .msi packages.



Task ***To determine the differences between two .msi packages:***

1. Open InstallShield MSI Diff.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.
3. Select the base .msi package, and then click **Open**. InstallShield MSI Diff opens the .msi database.
4. On the **File** menu, click **Compare To**. The **Open** dialog box opens.
5. Select the second .msi package, and then click **Open**.

InstallShield MSI Diff compares the data from the second .msi package with the data in the base .msi package, and displays the results.

Running InstallShield MSI Diff from the Command Line to Generate a Log File of Differences

You can run InstallShield MSI Diff from the command line to generate a log file that identifies the differences between two .msi, .msm, or .pcp files, or that shows the differences that are applied to a Windows Installer database by a transform (.mst) or patch file (.msp). You can also use this tool from the command line to generate a log file that identifies the differences between two InstallShield project files (.ism or .ise) that are saved in binary format.

InstallShield MSI Diff is installed to the following location:

***InstallShield Program Files Folder*\System\MsiDiff.exe**

To generate a log file that shows the differences, use the following syntax when you launch InstallShield MSI Diff from the command line:

```
msidiff.exe <BaseFile> <ComparisonFile> /out <diff.xml>
```

Following is a sample command line:

```
"C:\Program Files\InstallShield\2016\System\MsiDiff.exe" "C:\InstallShield 2016
    Projects\MyProject1.ism" "C:\InstallShield 2016 Projects\MyProject2.ism" /out "C:\Log File.xml"
```

Determining the Differences Between Two InstallShield Projects

You can use InstallShield MSI Diff to determine the differences between two InstallShield projects (.ism or .ise files) that are saved in binary format.



Task ***To determine the differences between two InstallShield projects:***

1. Open InstallShield MSI Diff.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.
3. Select one of the InstallShield project files, and then click **Open**. InstallShield MSI Diff opens the project file.
4. On the **File** menu, click **Compare To**. The **Open** dialog box opens.
5. Select the second InstallShield project file, and then click **Open**.

InstallShield MSI Diff compares the data from the second project with the data in the first project, and displays the results.

InstallShield MSI Query

InstallShield MSI Query lets you test SQL statements using the Windows Installer version of SQL before you run them in your build script. You can quickly see if a SQL statement is formatted correctly and view the results that it generates.

You can launch InstallShield MSI Query from the InstallShield Tools subfolder on the Windows Start menu.

To learn about valid SQL query strings for Windows Installer, see SQL Syntax in the Windows Installer Help Library.

Running a SQL Query for a Specific Windows Installer Database

You can use InstallShield MSI Query to test SQL statements using the Windows Installer version of SQL before you run them in your build script. You can quickly see if a SQL statement is formatted correctly and view the results that it generates.



Task ***To run a query:***

1. Open InstallShield MSI Query.
2. On the **File** menu, click **Open**. The **Open** dialog box opens.
3. Select the database (.msi or .msm file) that you want to use for your test, and then click **OK**. InstallShield MSI Query lists the name of the file in its title bar.
4. In the **Query** box, enter the query that you want to run.

   InstallShield MSI Query lets you enter one or more parameter values as question marks (?). If you include question marks in your query, InstallShield MSI Query displays an Arguments row below the Query box. Each question mark corresponds with a different Arguments box. Use the Arguments boxes to specify the replacement values.

   To learn about valid SQL query strings for Windows Installer, see SQL Syntax in the Windows Installer Help Library.
5. Click the **Execute** button.

InstallShield MSI Query runs the specified query for the selected database. If you would like to save any changes that occurred as a result of the query, click Save on the File menu.

InstallShield MSI Sleuth

InstallShield MSI Sleuth is a diagnostic tool that lets you view the current installed state of a target system. InstallShield MSI Sleuth displays a list of all of the .msi packages that are installed. You can click any .msi package in the list to see the tables in that database and all of the streamed files that are embedded within the database. This tool also helps you identify the installed product or products whose packages contain a specific component code.

You can launch InstallShield MSI Sleuth from the InstallShield Tools subfolder on the Windows Start menu.

Viewing Details about an .msi Package that Was Run on a Target System

You can use InstallShield MSI Sleuth to view details about an .msi package that was run a target system.



Task ***To view details about an .msi package:***

1. Open InstallShield MSI Sleuth.
2. On the **File** menu, click **Open Package**. The **Installed Packages** dialog box opens. It contains a list of all of the packages that have been installed on the target system.
3. Select the package that you want to analyze, and then click **OK**.

InstallShield MSI Sleuth displays details about the selected .msi package



Tip *• You can save the details in a .sleuth.xml and then reopen the file within MSI Sleuth. To save the details: On the File menu, click Save Package Details, and then specify a file name.*

To later open the .sleuth.xml file: On the File menu, click Open Package Summary. Select the file that you want to open.

Searching for All Installed Packages that Include a Specific Component

You can use InstallShield MSI Sleuth to identify all of the packages that have installed a component with a specified component code to resolve issues with files not being uninstalled. You can also determine whether a component was installed by multiple products.



Task ***To search for all installed packages that include a specific component:***

1. Open InstallShield MSI Sleuth.
2. On the **File** menu, point to **Open by Component Code**, and then point to **Component Code**. InstallShield MSI Sleuth changes this menu command to an edit field.
3. Enter the component code, and then press ENTER. Note that if you have the component code saved in the Clipboard, you can press CTRL+V to paste it in the menu command field. The **Installed Packages** dialog box opens.

The Installed Packages dialog box lists all of the installed packages that contain the specified component code. If you want to view a package, select it, and then click OK.

InstallShield MSI Grep

InstallShield MSI Grep searches a collection of .msi and .msm packages for specific text. You can narrow the search to show results for only certain tables or columns.

You can launch InstallShield MSI Grep from the InstallShield Tools subfolder on the Windows Start menu.

Searching .msi Packages for Specific Text

You can use InstallShield MSI Grep to search a collection of .msi and .msm packages for specific text. You can narrow the search to show results for only certain tables or columns.



Task ***To search one or more .msi and .msm packages for specific text:***

1. Open InstallShield MSI Grep.
2. In the **Find** box, enter the string for which you want to search.
3. To narrow your search, do one or more of the following:
   - In the **Table** box, specify the name of the database table that you want InstallShield MSI Grep to search through.

- In the **Column** box, specify the name of the database column that you want InstallShield MSI Grep to search through.
- To modify the list of folders that InstallShield MSI Grep should search, click the ellipsis button, and then select one or more folders.
- To search in all of the subfolders of the specified directories, select the **Recurse** check box.
- To limit the search to certain database types or file names, enter the file names in the **Files** box. Use an asterisk (*) as a wild-card character. If you specify multiple file names or patterns, separate each with a semicolon (;).

4. Click the **Search** button.

InstallShield MSI Grep displays the search results. You can click a search result, and then click a table from that search results to see details.

InstallShield Cabinet and Log File Viewer

Use the InstallShield Cabinet and Log File Viewer to do the following:

- View an InstallScript cabinet file (.cab) or InstallScript header file (.hdr), as well as their compressed files, registry entries, components, features, and other data. This tool also lets you extract files from the .cab file. This functionality applies to InstallScript and InstallScript Object projects.
- View InstallScript log files (.ilg) that are created by InstallScript installations. This tool enables you to see what your installation has recorded in the log file, which contains important uninstallation information in a binary format. This functionality applies to the following project types: InstallScript and InstallScript MSI.



Task ***To launch the InstallShield Cabinet and Log File Viewer from within InstallShield:***

On the **Tools** menu, point to **InstallScript**, and then point to **Cabinet/Log File Viewer**.



Tip • *You can also launch this tool from the InstallShield Tools subfolder on the Windows Start menu.*

To learn more, see the following sections of the documentation:

- Viewing InstallScript Cabinet Files (.cab) and Header Files (.hdr)
- Viewing InstallScript Log Files (.ilg)

Viewing InstallScript Cabinet Files (.cab) and Header Files (.hdr)



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The InstallShield Cabinet and Log File Viewer lets you open the **data1.cab** (the InstallScript cabinet file) and **data1.hdr** (the InstallScript header file that InstallShield creates for InstallScript and InstallScript Object projects). The InstallShield Cabinet and Log File Viewer also lets you extract files from those files.

When a **data1.cab** or **data1.hdr** file is open, the interface of the InstallShield Cabinet and Log File Viewer is divided into two panes. The left pane shows the **data1.cab** file or **data1.hdr** file as a directory structure, similar to Windows Explorer; the right pane shows information that is associated with the item that is selected in the left pane.

The folders in the right pane represent information that is used by your installation. In most cases, the folders do not contain files; when you select those types of folders, the viewer shows a grid of corresponding settings in the right pane. Click a folder to see the settings that are associated with that folder.

You cannot edit or alter **data1.cab** and **data1.hdr** files through the InstallShield Cabinet and Log File Viewer. However, you can save information about a **data1.cab** file or **data1.hdr** file and edit the resulting text file. For more information, see Generating a Report about an InstallScript .cab or .hdr File.

The InstallShield Cabinet and Log File Viewer includes an export mode that you can enable or disable, depending on how much information you want to see in the **data1.cab** or **data1.hdr** file:

- If you disable expert mode, the viewer shows only the files, registry entries, and other information that you added to your installation project.
- If you enable expert mode, the viewer shows all of the files, registry entries, and other information that you added to your installation project. The viewer also shows the files that InstallShield added to the **data1.cab** file; this includes files such as the InstallScript engine and the compiled script file (**Setup.inx**). In addition, the viewer shows InstallScript Objects that you included in the installation project.



Task ***To enable or disable expert mode in the InstallShield Cabinet and Log File Viewer:***

On the **View** menu, click **Expert Mode**.

Expert mode is enabled if the Expert Mode command has a check mark; this mode is disabled if the command does not have a check mark.

Overview of InstallScript .cab and .hdr Files



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

When you build an InstallScript installation, InstallShield may create an InstallScript cabinet file (.cab) called **data1.cab**, plus additional .cab files if necessary. InstallShield also creates an InstallScript header file called **data1.hdr**. Depending on your project's release settings, these files may be left uncompressed, or they may be or streamed into the **Setup.exe** file.

An InstallScript .cab file is a compressed file that contains the files that your installation installs on an end user's system. Cabinet files serve as a container for files that you directed your installation to install, either through InstallShield or through your script. Cabinet files also contain information and files about language and system dependencies. In addition, they store other files—such as the InstallScript engine—that InstallShield adds to your installation at build time.

The **data1.hdr** file is an InstallScript header file (.hdr) is a file that references the corresponding **data1.cab** file. It contains general information that you entered in your installation project. It also includes information about each of the files, components, features, and other data in your installation.

Important • *You can open* **data1.cab** *and* **data1.hdr** *files in the InstallShield Cabinet and Log File Viewer. Although InstallShield may create other .cab files such as* **data2.cab** *for your installation, the viewer can open only the* **data1.cab** *and* **data1.hdr** *files. Note that the* **data1.cab** *file, the* **data1.hdr** *file, and all other .cab files that are part of an InstallScript installation must be in the same folder when you are using the InstallShield Cabinet and Log File Viewer.*

Opening InstallScript .cab and .hdr Files

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

Important • *You can open* **data1.cab** *and* **data1.hdr** *files in the InstallShield Cabinet and Log File Viewer. Although InstallShield may create other .cab files such as* **data2.cab** *for your installation, the viewer can open only the* **data1.cab** *and* **data1.hdr** *files. Note that the* **data1.cab** *file, the* **data1.hdr** *file, and all other .cab files that are part of an InstallScript installation must be in the same folder when you are using the InstallShield Cabinet and Log File Viewer.*

Task ***To open a data1.cab or data1.hdr file in the InstallShield Cabinet and Log File Viewer:***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. Browse to the location that contains the file that you want to open, and then select that file.
3. Click the **Open** button.

The InstallShield Cabinet and Log File Viewer shows the following information for a **data1.hdr** or **data1.cab** file:

- **Features**—The viewer shows the settings that are configured for each feature in the installation. It also shows the components that are associated with each feature.
- **Components**—The viewer shows the settings that are configured for each component in the installation. Expand a component to see the files that are included in that component. You can extract any of the files.
- **Setup Types**—The viewer displays the settings that are configured for each setup type in the installation. This area of the viewer also shows the features that are associated with each setup type.
- **Explorer Shell**—The viewer lists each of the shell objects that you added to your project in InstallShield. For example, if you included any shortcuts or program folders in your installation, they are displayed in this area of the viewer.
- **Registry Entries**—The viewer lists all of the entries that are to be installed to the target system's registry.

- **Media Settings**—The viewer shows project settings that were configured in the General Information view. It also includes information about the version of InstallShield that created the file.

To view any of that information for a **data1.hdr** or **data1.cab** file, expand that node in the left pane of the InstallShield Cabinet and Log File Viewer, and click any of the nodes or subnodes.

Extracting a File from an InstallScript .cab or .hdr File



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

When a **data1.cab** or **data1.hdr** file is open in the InstallShield Cabinet and Log File Viewer, you can extract any of the files that are compressed into your installation's **cab** files and save them elsewhere.



Task ***To extract a file:***

1. In the left pane, expand the **Components** node, expand the component that contains the file that you want to extract, and then click the **Files** node under that component.
2. In the right pane, select the record for the file that you want to extract.
3. On the **Tools** menu, click **Extract File**. The **Save** dialog box opens.
4. Browse to the location where you want the InstallShield Cabinet and Log File Viewer to extract the file, and then click the **Save** button.



Note • *Media files and components may be password protected. If the entire media are password protected, you must enter the correct password before you can open the cabinet files that are associated with that media. If individual components are password protected, you must enter the correct password before you can extract the compressed files that are associated with that component.*

Generating a Report about an InstallScript .cab or .hdr File



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

You can use the InstallShield Cabinet and Log File Viewer to generate a text-file report that contains summary information about the files, components, features, and other data in the open **data1.cab** or **data1.hdr** file.



Task ***To generate a report about the open data1.cab or data1.hdr file:***

On the **Tools** menu, click **Report**.

The InstallShield Cabinet and Log File Viewer generates a report (.txt file) and opens it in a text editor such as Notepad.

Viewing InstallScript Log Files (.ilg)



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The InstallShield Cabinet and Log File Viewer enables you to view the log files that are created by your InstallScript or InstallScript MSI installations. The viewer lets you see what your installation has recorded in the log file, which contains important uninstallation information in a binary format.

When an InstallScript log file (.ilg) is open in the InstallShield Cabinet and Log File Viewer, the interface of the viewer is divided into two panes. The left pane shows log file entries divided into folders. When you select an item in the left pane, the right pane shows information about that item. For example, if you select a feature in the left pane, the right pane shows the State property, which indicates whether the feature is installed. If you select a File Operations item in the left pane, the right pane shows the information such as the files that were transferred to the target system, and whether those files were marked as permanent.

The <Support> folder and its subitems provide information about setup file operations. The <Data> folder and its subitems provide information about application file operations. The Disk<1> folder and its subitems provide information about the installation files that enable maintenance mode and uninstallation.

Overview of InstallScript Log Files



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Usually, an InstallScript log file is accessed only when your product is uninstalled or when an installation is aborted. Log files record the procedures that your installation performs when logging is enabled. These procedures may include the following:

- General information on the product being installed, such as the product name.
- The locations and values of an installed product's registry keys.
- Changes made to the system's file structure, including the files that were transferred, the folders that were created, and shortcuts or icons that were created.
- Information about the files that you added to your InstallShield project.

The name of the log file is **setup.ilg**. The installation creates it automatically at run time on a target system during normal file transfer. The default location for the log file is:

<PROGRAMFILES>\InstallShield Installation Information\{Product Code}

However, you can change the location by changing the value of the DISK1TARGET system variable before the Move Data event.

The InstallScript log file is a binary file. You can use the InstallShield Cabinet and Log File Viewer to see the contents of the log file.



Tip *• Logging of procedures is enabled by default. You can disable and re-enable logging by calling the Disable(LOGGING) and Enable(LOGGING) functions in your InstallScript code. For more information, see Disable and Enable.*

Opening InstallScript Log Files



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Use the InstallShield Cabinet and Log File Viewer to see the contents of an InstallScript log file (.ilg).



Task ***To open an InstallScript log file in the InstallShield Cabinet and Log File Viewer:***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. Browse to the location that contains the file that you want to open, and then select that file.
3. Click the **Open** button.

Searching InstallScript Log Files



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The InstallShield Cabinet and Log File Viewer lets you search for text in an InstallScript log file (.ilg).



Task ***To search for text in a log file:***

1. On the **Search** menu, **click** Find. The **Find** dialog box opens.
2. In the **Find What** box, type the text to be found.
3. Click **Find Next**. The first log entry (if any) that matches your search criteria is selected in the right pane.
4. To find the next item (if any) that matches your criteria, press the F3 key. Repeat this step as necessary.

Converting an InstallScript Log File to a Text File



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

Use the InstallShield Cabinet and Log File Viewer to create a text file (.txt) version of an InstallScript log file (.ilg).



Task ***To create a .txt file version of a log file:***

On the **Tools** menu, click **Report**.

The InstallShield Cabinet and Log File Viewer creates a .txt file version of the log file and opens it in a text editor such as Notepad.

Working with Windows Installer and Advanced UI or Suite/Advanced UI Properties

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

Note that some differences between properties in Windows Installer–based projects and that in Advanced UI or Suite/Advanced UI projects exist; these differences are noted where appropriate.

The Windows Installer, Advanced UI, and Suite/Advanced UI engines maintain global information using properties. The Property Manager view displays a list of installation properties that are used by the Windows Installer engine or the Advanced UI or Suite/Advanced UI engine at run time. You can modify, create, and delete installer properties in the Property Manager view. At build time, InstallShield writes the properties in the Property Manager view to installation that it creates.

For background information on property support in Windows Installer–based projects, see:

- Overview of Windows Installer Properties
- Windows Installer Property Reference

For background information on property support in Advanced UI and Suite/Advanced UI projects, see Advanced UI and Suite/Advanced UI Property Reference.

Overview of Windows Installer Properties

The built-in InstallShield properties are outlined in the Windows Installer Property Reference.

Property Types

There are four general types of Windows Installer properties:

- Public
- Private
- Restricted public
- Required

Note *• Some of these categories overlap. For example, the* ***ProductCode*** *property is a required private property.*

Public Properties

Public properties have names that contain only uppercase letters. For example, **INSTALLDIR** is a public property. Public properties can be specified at the command line used to launch the installation or chosen by using an authored user interface. If you are creating your own properties, use all uppercase letters if you want end users to have access to these properties.

In order to allow the end user or administrator to change the destination via the user interface or from the command line, the Directory Identifier for the component's destination must be a public property.



***Note** • Only public properties have their values preserved from an installation's user interface to the point where the installation is changing the target system. If you set the value of a property in a dialog box displayed to the end user, use a public property (for example, MY_PUBLIC_PROPERTY) if you want its value written to a file or to the registry.*

Private Properties

Private properties have at least one lowercase letter in their name and cannot be changed from the user interface. For example, ProgramFilesFolder is a private property. End users have no control over the values of private properties, as they cannot be set from the command line.

Restricted Public Properties

Restricted public properties allow network administrators to define public properties that can be changed only by a system administrator. This way, the administrator can change settings quickly without having to worry that other users on the network may tamper with the setup. Refer to Specifying that a Public Property Should Be a Restricted Public Property for more information.

Required Properties

The Windows Installer service relies on five properties that are required in every Windows Installer setup. By default, these properties are included in every installation you create using InstallShield. If you delete any of the following properties from your project, you need to reinsert them for your installation to function correctly.

- ProductCode
- ProductLanguage
- Manufacturer
- ProductVersion
- ProductName

Conditions

Many properties are not set until the installation is launched. These properties are populated with information from the target system. For example, the VersionNT property is not set until the installation is launched. This property is set to the version of Windows that the target machine is running, if the operating system is a Windows NT–based system. For example, the value of VersionNT is 600 on a Windows Vista system.

Properties set at run time can be used to create conditional installations. If you want your product to be installed only on Windows Vista, you can use conditional logic to check the end user's system, and install the product if all conditions are met.

Windows Installer Property Reference

Many properties can be set during an installation. The following tables list most of them with their functions and, in many cases, the syntax needed to put these properties into action. Many of the properties can be set from within InstallShield, while others are initialized by the Windows Installer service at run time.



***Note** • By default, Windows Installer writes the final value of every Windows Installer property contained in your project to a log file generated by launching MsiExec with the /L argument. Starting with Windows Installer version 2.0, you can prevent certain properties (such as those containing passwords) from being written to the log file by creating a property called **MsiHiddenProperties** in the Property Manager and setting its value to a semicolon-delimited list of property names. For more information about this property, see MsiHiddenProperties in the Windows Installer Help Library.*

Any of the properties that are with all-uppercase names can be set at the command line. Properties with all-uppercase names are called *public properties*. The following lists are organized according to the functions of the properties. Each table has a brief description of the kinds of properties it contains.

The following categories of Windows Installer properties are described in this topic. Click a link to go directly to a category.

- Special Folder and File Properties
- Feature Installation Properties
- Other Configurable Properties
- User-Supplied Information
- Properties for Creating Predetermined User Accounts
- Product-Specific Properties
- System Folders Set by the Installer
- Operating System Properties Set by the Installer
- Hardware Properties Set by the Installer
- PowerShell Properties
- Virtual Machine Properties
- Status Properties Updated by the Installer
- Date and Time Properties
- InstallScript Engine-Related Properties
- SQL-Related Properties
- MDAC Properties



***Note** • Do not confuse installer properties with path variables, which are surrounded by angle brackets (< >). While they both represent directories, you can use Windows Installer properties at run time, but path variables are used to point to source files only during setup design and at build time.*

Special Folder and File Properties

Special folder properties are those properties that define where files are stored or installed on the target system. File properties refer to specific files.

To use a folder property in your installation, enclose it in square brackets: []. For example, to install a component to a Bin folder under the default destination folder, enter **[INSTALLDIR]Bin** in the component's Destination Folder setting.

Table 9-1 • Folder Properties

Property Name	Description
INSTALLDIR	This property contains the default destination folder for the files in your features and components. For more information, see Setting the Default Product Destination Folder (INSTALLDIR). INSTALLDIR is available directly as a variable in your InstallScript code.
ISDEBUGLOG	To generate a log file for the feature prerequisites in the installation, set this property to the full path and file name for the log file. You can use directory properties and environment variables in the path. This property is used for troubleshooting feature prerequisites, and it is typically set through the command line when the `/debuglog` command-line parameter is used, as follows: `Setup.exe /debuglog"C:\Path\setup.log" /v"ISDEBUGLOG=prereq.log"`
ISPREREQDIR	This property identifies the path to the running InstallShield prerequisite. The path includes the final slash. You can use this property to refer to files within the InstallShield prerequisite—for example: **[ISPREREQDIR]MyConfigFile.ini** For more information, see Specifying Command-Line Parameters for an InstallShield Prerequisite.
SETUPEXEDIR	This property stores the path to **Setup.exe**. For example, if the path to **Setup.exe** is **C:\MySetups\MyApp\Setup.exe**, the value of **SETUPEXEDIR** is **C:\MySetups\MyApp**. For more information, see SETUPEXEDIR.
SETUPEXENAME	This property identifies the name of the setup launcher file that was created when the project was built. The installation updates the value of this property at run time if the setup launcher file was renamed. The following path identifies the full path to this file: **[SETUPEXEDIR]\[SETUPEXENAME]**
SourceDir	This property stores the folder where the running .msi file is located.

Table 9-1 • Folder Properties (cont.)

Property Name	Description
TARGETDIR	The **TARGETDIR** property is the location where the Windows Installer package is copied (and its data files uncompressed) during an Administrative installation. In an InstallScript MSI project, this value is represented by the InstallScript *MSI_TARGETDIR* variable.

Feature Installation Properties

The following section provides information on feature installation properties, with which the end user can specify how features should be installed.

Table 9-2 • Feature Installation Properties

Property Name	Description
ADDDEFAULT	The **ADDDEFAULT** property stores a list of features, separated by commas, that are to be installed in their default configuration. Users can install all features in their default configuration by setting the value of this property to **ALL**.
ADDLOCAL	This property stores a list of features, separated by commas, that are to be installed locally. Each feature has a Remote Installation property that determines whether features selected for installation are to be installed locally or run from the source medium.
ADDSOURCE	This property stores a list of features, separated by commas, that are to be run from the source medium. If this property is set to **ALL**, all the features will run from the source medium.
ADVERTISE	This property stores a list of features, separated by commas, that are to be advertised.
REINSTALL	This property stores a list of features, separated by commas, that are to be reinstalled. If **REINSTALL** is set to **ALL**, all of the features that are already installed on the user's system will be reinstalled.
REINSTALLMODE	This property contains a string of letters that specify the reinstall type. These options correspond to the values available for the Msiexec.exe `/f` command-line parameter. For more information on this property, see REINSTALLMODE Property in the Windows Installer Help Library. **REINSTALLMODE** is always set in conjunction with **REINSTALL**.

Table 9-2 • Feature Installation Properties (cont.)

Property Name	Description
ReinstallModeText	This property contains the reinstallation options that will be set when the user selects Repair in the MaintenanceType dialog. The value of ReinstallModeText is passed as an argument to the control event ReinstallMode, which accepts the same options that are available for the **REINSTALLMODE** property. ReinstallModeText is not a predefined Windows Installer property. Nonetheless, it is provided in the Property Manager for every new project, with a default value of *omus*, for use in the control event described above.
REMOVE	This property stores a list of features, separated by commas, that will be removed. If **REMOVE** is set to **ALL**, all features will be removed.
COMPADDLOCAL	This property stores a list of component IDs, separated by commas, that are to be installed locally. The feature for the component that takes up the least amount of disk space will be installed.
COMPADDSOURCE	This property stores a list of component IDs, separated by commas, that are to be run from the source medium. The feature for the component that takes up the least amount of disk space will be installed.
PATCH	When installing a patch, this property contains the full path to the patch package.

Other Configurable Properties

The following section contains information on various other configurable properties.

Table 9-3 • Additional Configurable Properties

Property Name	Description
ACTION	This property specifies which sequence to perform (installation, advertisement, or administration). Possible values are INSTALL, ADVERTISE, and ADMIN. The **ACTION** property is automatically set based on the command line used to launch the installation.

Table 9-3 • Additional Configurable Properties (cont.)

Property Name	Description
ALLUSERS	Specifies whether Windows Installer should attempt to perform a per-machine or per-user installation. If the value of **ALLUSERS** is set to 1, Windows Installer attempts a per-machine installation. For per-machine installations, configuration information such as shortcuts and registry entries are stored in the All Users profile: • On Windows Vista and later systems, if User Account Control is enabled and the user does not have administrative privileges, the user must be able to provide administrative credentials in order to install the product. • On other systems, if the user does not have administrative privileges, the installation displays an error message and exits.  ***Project*** *• This property is set to 1 by default in Basic MSI and InstallScript MSI projects. To learn more, see Per-User vs. Per-Machine Installations.* If the value of **ALLUSERS** is not set or it is an empty string (""), Windows Installer performs a per-user installation, and the configuration information is stored in the user's personal profile. If the value of **ALLUSERS** is set to 2, Windows Installer attempts to perform a per-machine installation on Windows Vista and later systems. On earlier platforms, if the user has administrative privileges, Windows Installer attempts to perform a per-machine installation; otherwise, Windows Installer performs a per-user installation.  ***Project*** *• This property is set to 1 in InstallScript MSI projects to help avoid* ***ALLUSERS****-related issues when the installation is run silently. The property can be overridden with the* ***SdCustomerInformation*** *or* ***SdCustomerInformationEx*** *dialog functions in InstallScript MSI projects.*  ***Note*** *•* ***ALLUSERS*** *has no effect on InstallScript variables like* `FOLDER_PROGRAMS`*.* ***ALLUSERS*** *has no effect until the Execute sequence runs.*
ARPAUTHORIZEDCDFPREFIX	This property stores the URL for the update channel of the application.

Table 9-3 • Additional Configurable Properties (cont.)

Property Name	Description
ARPCOMMENTS	This property contains the comments about this product that are displayed in Add or Remove Programs. This value is set in the ARP Comments setting in the General Information view. You should provide a string entry to facilitate globalizing your installation.
ARPCONTACT	This property contains support contact information, such as an email address or a telephone number. This value is set in the Support Contact setting in the General Information view. You should provide a string entry to facilitate globalizing your installation.
ARPINSTALLLOCATION	This property stores the fully qualified path to the application's primary folder. You can set **ARPINSTALLLOCATION** to the value of **INSTALLDIR** with a custom action of type "Set a property".
ARPNOREPAIR	If this property is set to 1, no Repair button will be displayed in the Programs Wizard.
ARPREADME	Holds the fully qualified path or the URL for the product's Readme file. This value is set in the Read Me setting in the General Information view. You should provide a string entry to facilitate globalizing your installation.
ARPSIZE	This property stores the estimated size, in kilobytes, of the application.
ARPSYSTEMCOMPONENT	Set this property to 1 to keep your program from appearing in Add or Remove Programs.
ARPURLINFOABOUT	This property stores the URL for the application's home page or the publisher's home page. This value is set in the Publisher/Product URL setting in the General Information view. You should provide a string entry to facilitate globalizing your installation.
ARPURLUPDATEINFO	This property stores the URL for update information on your application. This value is set in the Product Update URL setting in the General Information view. You should provide a string entry to facilitate globalizing your installation.

Table 9-3 • Additional Configurable Properties (cont.)

Property Name	Description
ARPNOMODIFY	Setting this property prevents the product from being modified from Add or Remove Programs.
ARPNOREMOVE	Setting this property prevents the product from being removed through Add or Remove Programs.
AVAILABLEFREEREG	This property allows you to set the amount of additional free registry space, in kilobytes, required by your application.
CCP_DRIVE	This property holds the root path on the installation disk for any of the qualifying products for a competitive upgrade.
DISABLEADVTSHORTCUTS	Setting this property disables the creation of advertised shortcuts.
DISABLEMEDIA	This setting prevents the installer from adding media information to the source list.
DISABLEROLLBACK	Set this property to 1 to stop the installer from creating a rollback script that will save copies of changed or deleted files during the installation process.
EXECUTEACTION	This property sets the top-level action initiated by the ExecuteAction action.
EXECUTEMODE	This property sets the mode of execution performed by the installer. The value None means that no changes are made to the system. The default value, Script, means that all changes made to the system are run through a script execution.
INSTALLLEVEL	This property holds a value that corresponds to the values of the features. If the level of features to be installed is less than or equal to the **INSTALLLEVEL** property, then a feature is installed. This is primarily used for different setup types, such as Typical or Custom.
LOGACTION	List of action names, separated by semicolons, that will be logged.

Table 9-3 • Additional Configurable Properties (cont.)

Property Name	Description
MSIINSTALLPERUSER	This property indicates that the Windows Installer should install the package for only the current user: • If the **ALLUSERS** property is set to 2 and **MSIINSTALLPERUSER** is set to an empty string (""), the Windows Installer performs a per-machine installation. • If the **ALLUSERS** property is set to 2 and **MSIINSTALLPERUSER** is set to 1, the Windows Installer performs a per-user installation. This property is available with Windows Installer 5 and on Windows 7 or Windows Server 2008 R2. Earlier versions of Windows Installer and Windows ignore this property. For more information, see the following: • Per-User vs. Per-Machine Installations • MSIINSTALLPERUSER Property on the MSDN Web site
Privileged	This property will run an installation with elevated privileges if the user is an administrator or if the application is an administrator-assigned application.
PROMPTROLLBACKCOST	This property specifies what will happen if there is insufficient disk space to continue the installation. Depending on the user interface level, the rollback could happen automatically, without any input from the user, or it could ask the user to continue with rollback disabled.
PRIMARYFOLDER	The folder that you specify with this property will be the "primary" folder for the installation. The path to this folder will be used to determine the values for the **PrimaryVolumePath** property, the **PrimaryVolumeSpaceAvailable** property, the **PrimaryVolumeSpaceRequired** property, and the **PrimaryVolumeSpaceRemaining** property.
REBOOT	This property allows you to force or suppress a reboot after the installation completes. Possible values are: • **F**—To force a reboot when your installation is complete. • **S**—To suppress any reboot except one caused by the ForceReboot action. • **R**—To suppress any reboot caused by Windows Installer actions.

Table 9-3 • Additional Configurable Properties (cont.)

Property Name	Description
ROOTDRIVE	In Administration mode this property sets the default drive to the first writable network drive found. In all other modes, this property sets the default drive to the writable local drive with the most disk space available.
SCRIPTFILE	This property defines the location of the script file that contains all operations executed during the installation.
SEQUENCE	This property specifies an .msi database table that lists the order in which the actions in the table will run.
SHORTFILENAMES	In Administration mode, this property may be set to indicate that only short file names should be used.
TRANSFORMS	This property stores a list of transforms to be applied to an MSI database. These transforms can be set only in Installation and Advertisement mode. The syntax for this property, if you are applying a transform to two tables, is as follows: c:\transforms\trans1;:trans2. Note that transforms are applied in the order that they appear in the string.
TRANSFORMSATSOURCE	By setting this property to 1, you are specifying the installer to look for the transforms at the installation source.
LIMITUI	Setting this property limits the user interface level at basic. This is useful if you do not create a custom user interface to interact with the installer's built-in UI.
DefaultUIFont	This property should be set to one of the predefined styles found in the TextStyle table in order to specify your default font. If this property is not set, the installer will use the system font, which may disrupt your formatting.

User-Supplied Information

The following section contains information about input taken from the end user. Such input can include the end user's name, company, or language.

Table 9-4 • User-Supplied Information

Property Name	Description
AdminProperties	**AdminProperties** holds a list of properties set during an administration installation. These properties can be external (user name) or they can be internal (other properties on this page).

Table 9-4 • User-Supplied Information (cont.)

Property Name	Description
COMPANYNAME	This property stores the organization name for the end user performing the installation. This information is taken from the **Customer Information** dialog box (for Basic MSI projects), or from the SdCustomerInformation or SdCustomerInformationEx dialogs (for InstallScript MSI projects).
ISX_SERIALNUM	This property stores the serial number that the end user enters in the Serial Number field on the CustomerInformation dialog.
UserLanguageID	This property retains the default language identifier for the end user.
USERNAME	This property stores the name of the end user performing the installation, which is taken from the **Customer Information** dialog (for Basic MSI projects), or from the **SdCustomerInformation** or **SdCustomerInformationEx** dialogs (for InstallScript MSI projects).
ProductLanguage	This property stores the numeric language ID for the product.

Properties for Creating Predetermined User Accounts

The following table describes the properties that let you create one or more Windows user accounts without using logon dialogs.

Table 9-5 • Properties for Creating Predetermined User Accounts

Property Name	Description
ISNetApiLogonUsername	Set the value of this property to the user account that you want the installation to create. Use either of the following formats: • *MachineName\UserName* • *DomainName\UserName* For more information, see Creating Predetermined User Accounts at Run Time.
ISNetApiLogonGroup	Set the value of this property to the group to which you want the user account to belong. For more information, see Creating Predetermined User Accounts at Run Time.
ISNetApiLogonPassword	**ISNetApiLogonPassword**—Set the value of this property to the password that you want to be configured for the user account. For more information, see Creating Predetermined User Accounts at Run Time.

Product-Specific Properties

Information on product-specific properties that can be set in the Property table is listed below. Examples of these types of properties include technical support numbers, product name, and serial number.

Table 9-6 • Product-Specific Properties

Property Name	Description
ARPHELPLINK	This property retains the Internet address for technical support. This value is set in the Support URL setting in the General Information view. You should provide a string entry to facilitate globalizing your installation.
ARPHELPTELEPHONE	This property retains your technical support telephone numbers. This value is set in the Support Phone Number setting in the General Information view. You should provide a string entry to facilitate globalizing your installation.
ProductCode	The ProductCode is the GUID for this particular version of the product. This ID must be different for different language versions and different release versions. This property is set in the General Information view.
ProductName	This property stores the name of the product—for example, InstallShield. This property is set in the General Information view.
ProductState	The installer sets this property to the installed state of the product. This property can hold one of four numeric values: **-1**—The product has not been installed or advertised. **1**—The product has been advertised, but not installed. **2**—The product has been installed for another user. **5**—The product has been installed and is available to the current user.
ProductVersion	The **ProductVersion** property stores the major, minor, and build version numbers in the format *AA.BB.CCCC*. This property is set in the General Information view.
Manufacturer	Stores the name of the product manufacturer. This value is set in the **Publisher** setting in the General Information view. You should provide a string entry to facilitate globalizing your project.
DiskPrompt	This property holds a string which is displayed by a message box prompting for a disk. You should also include empty text for additional information printed on the disk label, as in "Disk 1".

Table 9-6 • Product-Specific Properties (cont.)

Property Name	Description
DiskSerial	The **DiskSerial** property should be set to the internal serial number for this release.
ComponentDownload	This property retains the URL for downloading a product by its string identifier (GUID).
LeftUnit	This property places units to the left of the number. This is necessary for languages that require this structure.
UpgradeCode	This is a GUID used to search for a related set of products that are already installed.
IsAdminPackage	This property is set to 1 if the current installation package was created through an administrative installation. You can use this property to detect post-administrative installations.

System Folders Set by the Installer

The following properties hold the fully qualified path to many of the folders on the end user's system. Many of these properties can be used directly in your script, without having to call MsiGetProperty.

Table 9-7 • System Folder Properties

Property Name	Description
AppDataFolder	This property holds the full path to the current user's Application Data folder.
CommonAppDataFolder	This property holds the full path to the All Users Application Data folder.
CommonFilesFolder	The value of this property is the full path to the 32-bit Common Files folder.
CommonFiles64Folder	The value of this property is the full path to the 64-bit Common Files folder. This property requires Windows Installer version 2.0.
DesktopFolder	This property is used to hold the full path to the Desktop folder for the current user. If the **ALLUSERS** property is set, the DesktopFolder property should hold the full path to the All Users desktop folder.
FavoritesFolder	The **FavoritesFolder** property retains the full path to the Favorites folder for the current user.
FontsFolder	This property holds the full path to the Fonts folder.

Table 9-7 • System Folder Properties (cont.)

Property Name	Description
PersonalFolder	This property holds the full path to the current user's Personal folder.
ProgramFilesFolder	This property holds the full path to the current user's Program Files folder.
ProgramFiles64Folder	This property holds the full path to the current user's 64-bit Program Files folder. This property requires Windows Installer version 2.0.
ProgramMenuFolder	This property is used to hold the full path to the **Program** menu for the current user. If the **ALLUSERS** property is set, the **ProgramMenuFolder** property should hold the full path to the All Users Programs menu.
SendToFolder	This property holds the full path to the current user's SendTo folder.
StartMenuFolder	This property is used to hold the full path to the Start menu folder for the current user. If the **ALLUSERS** property is set, the **StartMenuFolder** property should hold the full path to the All Users Start Menu folder.
StartupFolder	This property is used to hold the full path to the Startup folder for the current user. If the **ALLUSERS** property is set, the **StartupFolder** property should hold the full path to the All Users Startup menu.
SystemFolder	This property holds the full path to the 32-bit System folder.
System64Folder	This property holds the full path to the 64-bit System folder. This property requires Windows Installer version 2.0.
TempFolder	This property holds the full path to the Temp folder.
TemplateFolder	This property holds the full path to the current user's Template folder.
WindowsFolder	This property holds the full path to the user's Windows folder.
WindowsVolume	This property is set to the drive where Windows is installed.

Operating System Properties Set by the Installer

The following properties are set by the installer at run time. They refer to environment variables on the target system.

Table 9-8 • Operating System Properties

Property Name	Description
AdminUser	This property is set by the installer at installation and is only set if the user has administrative privileges.
ComputerName	This property stores the name of the computer that the installation is running on. It is set by a call to the Windows API **GetComputerName** at the initialization of the installer.
LogonUser	This property stores the name of the user performing the installation. It is set by a call to the Windows API **GetUserName**.
OLEAdvtSupport	This property is set by the installer during initialization.
ServicePackLevel	If an operating system service pack is installed, this property stores the numeric value for that update.
SharedWindows	This property is set when **Shared Windows** is being used on the target system.
ShellAdvtSupport	This property is set by the installer during initialization if the target system supports feature advertisement. This property is automatically set on Windows 98 or later, or on earlier systems if Internet Explorer 4.01 is installed.
SystemLanguageID	This property stores the default language identifier for the target system. The value is defined by the installer by calling GetSystemDefaultLangID at initialization.
TerminalServer	This property is set by the installer at initialization if the target system is a server with Windows Terminal Server.
TTCSupport	This property is set by the installer at initialization if the target system supports true type font collections (TTC). The following systems support TTC: JPN - 932, Taiwan - 950, China - 936, Korea - 949, Hong Kong - 950.
Version9X	This property stores the version number of Windows 95 and 98 operating systems as an integer. The following formula is used to determine this integer: (MajorVersion * 100) + MinorVersion. On Windows 95, Version9X is set to 400, on Windows 98 it is set to 410, and on Windows Millennium Edition it is set to 490. Version9X is not set on Windows NT-based systems.

Table 9-8 • Operating System Properties (cont.)

Property Name	Description
VersionDatabase	This property stores a version number of the database used during the installation.
VersionNT	This property stores the version number of Windows NT-based operating systems as an integer. The following formula is used to determine this integer: (MajorVersion * 100) + MinorVersion. Refer to the Windows Installer Help Library to learn the VersionNT property for a specific operating system.
VersionNT64	This property stores the version number of a Windows NT-based operating system as an integer on 64-bit systems only. The following formula is used to determine this integer: (MajorVersion * 100) + MinorVersion. This property requires Windows Installer version 2.0.
WindowsBuild	This property stores the build number for the operating system being run.
MsiNTProductType	This property stores the type of NT operating system being run on the target machine. This property requires Windows Installer version 2.0.
MsiNTSuiteBackOffice	This property is set to 1 if Microsoft BackOffice components are installed. In all other cases this property is not set. This property requires Windows Installer version 2.0.
MsiNTSuiteDataCenter	This property is set to 1 if Windows 2000 Datacenter Server is installed. In all other cases this property is not set.
MsiNTSuiteEnterprise	This property is set to 1 if Windows 2000 Advanced Server is installed. In all other cases this property is not set.
MsiNTSuiteEnterprise	This property is set to 1 if Windows 2000 Advanced Server is installed. In all other cases this property is not set.
MsiNTSuiteSmallBusiness	This property is set to 1 if Microsoft Small Business Server is installed. In all other cases this property is not set.
MsiNTSuiteSmallBusinessRestricted	This property is set to 1 if Microsoft Small Business Server is installed with the restrictive client license. In all other cases this property is not set.
MsiNTSuitePersonal	This property is set to 1 if the operating system is Workstation Personal. In all other cases this property is not set.
MsiNetAssemblySupport	This property is set if the operating system supports .NET Framework assemblies. In all other cases this property is not set.

Table 9-8 • Operating System Properties (cont.)

Property Name	Description
MsiWin32AssemblySupport	This property is set if the operating system supports Win32 assemblies. In all other cases this property is not set.

Hardware Properties Set by the Installer

The following properties are set by the installer at run time and store settings on certain hardware profiles for the end user's system.

Table 9-9 • Hardware Properties

Property Name	Description
Alpha	This property stores the numeric value of the processor level, and it is only defined if the setup is running on an Alpha processor. (This property is supported only with Windows Installer version 1.0.)
BorderSide	This property sets the pixel width of the side window borders.
BorderTop	This property sets the pixel width of the top window border.
CaptionHeight	This property sets the pixel height of the caption area.
ColorBits	This property stores the number of adjacent color bits for each pixel (that is, the color depth of the user's monitor). For example, if the user's monitor is using 256 colors, ColorBits is set to 8.
Intel	This property stores the numeric value of the processor level, and it is defined only if the setup is running on an Intel 32-bit processor.
Intel64	This property stores the numeric value of the processor level, and it is defined only if the setup is running on an Intel 64-bit processor. This property requires Windows Installer version 2.0.
PhysicalMemory	This property stores the installed amount of physical memory, in megabytes.
ScreenX	This property defines the width of the screen, in pixels.
ScreenY	This property defines the height of the screen, in pixels.
TextHeight	This property sets the height of text characters.
VirtualMemory	The amount of available page file space, in megabytes, is stored in this property.

PowerShell Properties

The following PowerShell-related properties are available at run time:

Table 9-10 • PowerShell Properties

Property Name	Description

IS_CLR_VERSION	This property identifies a semicolon-delimited list of .NET Framework versions that a custom action should attempt to load to run your managed code and PowerShell scripts. In most scenarios, this property is not set in the installation package. It is set at the command line. To specify that version 1.1 is required, use the following command-line parameter:
	`IS_CLR_VERSION=v1.1.4322`
	Note that the complete version number of the .NET Framework should be specified for the property value. If more than one version is acceptable, you can specify a semicolon-delimited list of versions. The first one that can be loaded is used. For the following example command-line parameter, the action attempts to load version 2.0. If that version is not present, the action attempts to load version 1.1. If version 1.1 is not present, the action fails.
	`IS_CLR_VERSION=v2.0.50727;v1.1.4322`
	To specify that the action should attempt to load whatever is the latest version of the .NET Framework that is installed if none of the specified versions are installed, add a semicolon to the end of the property value, as shown in the following example:
	`IS_CLR_VERSION=v2.0.50727;v1.1.4322;`
	The semicolon at the end of the property value also indicates that if none of the specified versions are present but a version of the .NET Framework is already loaded, the action uses the currently loaded version, even if it is not the latest version that is installed.
	If issues with the action occur at run time because of .NET Framework version mismatches, you may want to instruct end users to set the **IS_CLR_VERSION** property at the command line when they run your installation.
	To learn more, see:
	• Calling a PowerShell Custom Action
	• Run-Time Requirements for Managed-Code Custom Actions

POWERSHELLVERSION	The predefined PowerShell system search sets the value of this property if PowerShell is installed.

Virtual Machine Properties

To use Windows Installer properties to detect the presence of a virtual machine and determine the type of virtual machine, you first need to create a custom action that calls the ISDetectVM function in the **SetAllUsers.dll** file. For instructions, see Detecting Whether the Installation Is Being Run on a Virtual Machine.

The custom action sets the following Windows Installer properties at run time:

Table 9-11 • Virtual Machine Properties

Property Name	Description
IS_VM_DETECTED	If the value of this property is 1, one of the following virtual machine environments is present: • Microsoft Hyper-V • A VMware product such as VMware Player, VMware Workstation, or VMware Server • Microsoft Virtual PC If the value is not set, no virtual machine has been detected.
IS_VM_TYPE	This property indicates the type of virtual machine that is present on a target system. Available values are: • **0**—No virtual machine has been detected. • **1**—The installation is running on a VMware product such as VMware Player, VMware Workstation, or VMware Server. • **2**—The installation is running on a Microsoft Hyper-V machine. • **3**—The installation is running on a Microsoft Virtual PC machine. • **4**—The type of virtual machine is not known.

Status Properties Updated by the Installer

The following properties are set by the installer at run time. These properties identify the status of the installation.

Table 9-12 • Status Properties

Property Name	Description
AFTERREBOOT	This property is set to 1 by the installer after a reboot triggered by the ForceReboot action.
CostingComplete	This property is set to 1 as soon as costing has begun and is set to 0 when costing is complete.
RollbackDisabled	The installer sets the **RollbackDisabled** property whenever rollback has been disabled. This property is not set by default.
Installed	This property determines if the product is already installed.
OutOfDiskSpace	This property is set to True if any of the drives that are targets for the install do not have enough disk space. Otherwise, it is set to False.

Table 9-12 • Status Properties (cont.)

Property Name	Description
OutOfNoRbDiskSpace	This property is set to True if any disk targeted during an installation does not have enough free disk space and if rollback capability is turned off. It is set to False if all of the target disks have sufficient space.
Preselected	This property determines if features have been preselected, and if so, does not display the selection dialog.
PrimaryVolumePath	The installer sets this property to the path specified in the PRIMARYFOLDER property.
PrimaryVolumeSpaceAvailable	This installer sets this property to a string representing the total number of bytes, in units of 512, available on the volume specified by the **PRIMARYFOLDER** property.
PrimaryVolumeSpaceRequired	This property stores a string representing the total amount of disk space required, in bytes (expressed in units of 512 bytes), by the currently selected features.
PrimaryVolumeSpaceRemaining	The installer sets this property to a string representing the number of remaining bytes available on the system, in units of 512, if all selected features were to be installed.
Resume	This property stores the text string displayed to the user when an installation is resumed from a suspended setup.
UpdateStarted	This property is set once changes to the system have taken place as a result of the installation process.
ReplacedInUseFiles	This property is set if a file that is currently in use is overwritten. Custom actions that check to see if a reboot is required will use this property.
NOUSERNAME	Setting this property to 1 stops the installer from setting the **USERNAME** property. By default, this property is not set and the **USERNAME** property is set from the registry.
NOCOMPANYNAME	Setting this property to 1 stops the installer from setting the **COMPANYNAME** property. By default, this property is not set and the **COMPANYNAME** property is set from the registry.

Date and Time Properties

Table 9-13 • Date and Time Properties

Property Name	Description
Date	This property holds the current date.
Time	This property holds the current time.

InstallScript Engine-Related Properties

These properties contain information related to the InstallScript engine.

Table 9-14 • InstallScript Engine-Related Properties

Property Name	Description
STANDARD_USE_SETUPEXE	 ***Project*** • *This property applies to InstallScript MSI projects.* InstallScript MSI projects must be run by launching **Setup.exe**. This localizable property holds the message that is displayed to an end user who launches your .msi database directly, instead of launching **Setup.exe**.

SQL-Related Properties

These properties contain data related to SQL connections and SQL scripts that are configured in the SQL Scripts view.

Table 9-15 • InstallScript Engine-Related Properties

Property Name	Description
IS_SQLSERVER_ALIAS_ONLY	Specify to show only the SQL Server aliases in the SQL Server browse combo box and list box controls. To learn more, see Overriding the Default SQL Run-Time Behavior.
IS_SQLSERVER_AUTHENTICATION	This property identifies the type of authentication that you want to use to connect to the specified catalog. The default property is **IS_SQLSERVER_AUTHENTICATION**. The following numbers are valid property values: • **0**—Windows authentication credential of the current user • **1**—Server authentication To learn more, see Using Windows Installer Properties for SQL Login Settings.

Table 9-15 • InstallScript Engine-Related Properties (cont.)

Property Name	Description
IS_SQLSERVER_CONNECTIONS_TO_VALIDATE	Overrides the connections that will be tested when clicking the Next button on the SQLLogin dialog. To learn more, see Overriding the Default SQL Run-Time Behavior.
IS_SQLSERVER_CXNS_ABSENT_FROM_INSTALL	Specifies one or more SQL connections that should be skipped during installation or uninstallation. To learn more, see Overriding the Default SQL Run-Time Behavior.
IS_SQLSERVER_DATABASE	This property identifies the name of the SQL catalog to which you want to create a connection during the installation. To learn more, see Using Windows Installer Properties for SQL Login Settings.
IS_SQLSERVER_DO_NOT_USE_REG	Specifies not to use the stored SQL Server login information written in the registry. To learn more, see Overriding the Default SQL Run-Time Behavior.
IS_SQLSERVER_LOCAL_ONLY	Specifies to show only the local SQL Server in the SQL Server browse combo box and list box controls. To learn more, see Overriding the Default SQL Run-Time Behavior.
IS_SQLSERVER_PASSWORD	This property identifies the password that should be used for server authentication. To learn more, see Using Windows Installer Properties for SQL Login Settings.
IS_SQLSERVER_REMOTE_ONLY	Specifies to show only the remote SQL Servers in the SQL Server browse combo box and list box controls. To learn more, see Overriding the Default SQL Run-Time Behavior.
IS_SQLSERVER_SERVER	This property identifies the name of the target server instance (for Microsoft SQL Server and MySQL) or the connect URL string or local net service name (for Oracle). To learn more, see Using Windows Installer Properties for SQL Login Settings.
IS_SQLSERVER_USERNAME	This property identifies the login ID that should be used for server authentication. To learn more, see Using Windows Installer Properties for SQL Login Settings.

MDAC Properties

These properties apply to version checking for MDAC.

Table 9-16 • MDAC Properties

Property Name	Description
ISINSTALL_MDAC_BYVERSION	Create and set this property to Yes if you want InstallShield to perform default version checking.
ISINSTALL_MDAC_SKIP	Create and set this property in the Project Manager if you are performing your own version checking and do not want InstallShield to install MDAC.

SETUPEXEDIR



Project • *The following project types support the* ***SETUPEXEDIR*** *property:*

- *InstallScript MSI*
- *Basic MSI*

To determine the location from which an InstallScript installation was run, use SRCDIR or PACKAGE_LOCATION.

SETUPEXEDIR is a property that contains the path to **Setup.exe**. For example, if the path to **Setup.exe** is **C:\MySetups\MyApp\Setup.exe**, the value of **SETUPEXEDIR** is **C:\MySetups\MyApp**.

Using SETUPEXEDIR

SETUPEXEDIR is an alternative to the directory identifier **SourceDir**. A potential problem with using **SourceDir** is that it points to the location of the running .msi package. In the case of a compressed installation, the .msi package is streamed to a temporary location and run from there. Because of this, the value of **SourceDir** will be some temporary location on the end user's machine, which might not be the value that you want.

Limitations of SETUPEXEDIR

There are two limitations to using **SETUPEXEDIR**:

- **SETUPEXEDIR** is set by **Setup.exe**. If the end user runs the .msi package directly, **SETUPEXEDIR** is not set. To account for this, you could have a dual implementation in your installation—one that uses **SETUPEXEDIR** and one that uses **SourceDir**. You could test for the existence of **SETUPEXEDIR** and, if it does not exist, you could conditionally use your **SourceDir** implementation.
- **SETUPEXEDIR** might not be set at uninstallation. If the end user triggers the uninstallation by running **Setup.exe**, **SETUPEXEDIR** is set. If they run the uninstallation from Add or Remove Programs, **SETUPEXEDIR** is not set.



Note • *In an uncompressed installation,* ***SourceDir*** *and* ***SETUPEXEDIR*** *have the same value.*

Advanced UI and Suite/Advanced UI Property Reference

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

A number of properties are set during Advanced UI and Suite/Advanced UI installations. You can set the values of some of the properties from within InstallShield, while others are initialized by the Advanced UI or Suite/Advanced UI engine at run time.

Unlike Basic MSI projects, Advanced UI and Suite/Advanced UI projects do not distinguish between public properties and private properties. However, capitalization must be maintained; if you use mixed uppercase and lowercase for the name of the Advanced UI or Suite/Advanced UI property, ensure that you use the same capitalization every time that you refer to that property in your project.

Note that if a property that you define in the Property Manager view is set to a value that references a second property, and if you want the second property to be resolved in the first property's value, you may need to select the Formatted check box for the first property. To learn more, see Property Manager View.

The following categories of built-in Advanced UI and Suite/Advanced UI properties are available:

- Special Advanced UI and Suite/Advanced UI Properties
- Advanced UI and Suite/Advanced UI Folder Properties
- System Folder Properties

Note *• Do not confuse Advanced UI and Suite/Advanced UI properties with path variables, which are surrounded by angle brackets (< >). While they both may represent directories, Advanced UI and Suite/Advanced UI properties are evaluated at run time, but path variables are used to point to source files only during installation authoring and at build time.*

Special Advanced UI and Suite/Advanced UI Properties

The following properties indicate various miscellaneous information for the Advanced UI and Suite/Advanced UI installation.

Table 9-17 • Special Advanced UI and Suite/Advanced UI Properties

Property	Description

IS_CLR_VERSION	This property identifies a semicolon-delimited list of .NET Framework versions that a Suite/Advanced UI action should attempt to load to run your managed code and PowerShell scripts. In most scenarios, this property is not set in the installation package. It is set at the command line. To specify that version 1.1 is required, use the following command-line parameter:
	`IS_CLR_VERSION=v1.1.4322`
	Note that the complete version number of the .NET Framework should be specified for the property value. If more than one version is acceptable, you can specify a semicolon-delimited list of versions. The first one that can be loaded is used. For the following example command-line parameter, the action attempts to load version 2.0. If that version is not present, the action attempts to load version 1.1. If version 1.1 is not present, the action fails.
	`IS_CLR_VERSION=v2.0.50727;v1.1.4322`
	To specify that the action should attempt to load whatever is the latest version of the .NET Framework that is installed if none of the specified versions are installed, add a semicolon to the end of the property value, as shown in the following example:
	`IS_CLR_VERSION=v2.0.50727;v1.1.4322;`
	The semicolon at the end of the property value also indicates that if none of the specified versions are present but a version of the .NET Framework is already loaded, the action uses the currently loaded version, even if it is not the latest version that is installed.
	If issues with the action occur at run time because of .NET Framework version mismatches, you may want to instruct end users to set the **IS_CLR_VERSION** property at the command line when they run your installation.

ISFeatureInstall	This property stores a comma-delimited list of feature names that are defined in the Advanced UI or Suite/Advanced UI **Setup.xml** file for command-line feature selection that will be selected for installation.
ISFeatureRemove	This property stores a comma-delimited list of feature names that are defined in the Advanced UI or Suite/Advanced UI **Setup.xml** file for command-line feature selection that will be selected for removal.

Table 9-17 • Special Advanced UI and Suite/Advanced UI Properties (cont.)

Property	Description
ISHiddenProperties	This property stores a semicolon-delimited list of case-sensitive property names whose values you do not want to be written to the debug log files. This property enables you to prevent properties that contain passwords and other sensitive information from being logged. Note that **ISHiddenProperties** is useful for prevention of logging for values in the following scenarios: • The value of the property changes. • The end user sets the value of the property through command line when launching the Advanced UI or Suite/Advanced UI **Setup.exe** file. • The property is configured through a command line that the Advanced UI or Suite/Advanced UI installation passes to a package. This is configurable in the Packages view, on the Common tab, in the Operation area. If the property is logged through ISuiteExtension::LogInfo, the Suite engine cannot prevent the logging. Therefore, any code that you create to write a property value to the log file should read **ISHiddenProperties** to see whether the property value should be logged.
ISInstallMode	This read-only property stores a value that indicates the mode in which the Advanced UI or Suite/Advanced UI installation is running. Possible values are: • **0**—First-time installation • **1**—Maintenance/UI maintenance mode selection • **2**—Maintenance/modify • **3**—Maintenance/remove • **4**—Maintenance/repair • **5**—Stage only (The installation is run in stage mode, in which the Advanced UI or Suite/Advanced UI installation is staged—or uncompressed and copied to a specified location. Note that none of the packages in the installation are run in this mode.)
ISInstallProgress	This property stores the percentage of how much of the entire Advanced UI or Suite/Advanced UI installation has been completed. It is typically used for a progress circle control on the InstallationProgress wizard page.
ISInstallStatus	This property stores progress messages from the Advanced UI or Suite/Advanced UI installation.
ISOnRebooted	If the Advanced UI or Suite/Advanced UI installation is resuming after a reboot, this property is set to True.

Table 9-17 • Special Advanced UI and Suite/Advanced UI Properties (cont.)

Property	Description
ISParcelProgress	This property stores the percentage of how much of the currently running package installation has been completed. It is typically used for the Interior Property setting of the progress circle control.
ISPassword	If you configure your Advanced UI or Suite/Advanced UI release to be password protected, the Advanced UI or Suite/Advanced UI user interface sets the *ISPassword* property. When the password is set and a password is populated in the Advanced UI or Suite/Advanced UI **Setup.xml** file, the value of this property is validated against the password from **Setup.xml**. To learn more, see Password-Protecting an Advanced UI or Suite/Advanced UI Installation.
ISPerformDelayedReboot	This property is set to True by the Advanced UI or Suite/Advanced UI user interface if the target system should be restarted when the Advanced UI or Suite/Advanced UI installation exits.
ISParcelStatus	This property stores progress messages from the currently running package.
ISRootStagePath	This property is used when the installation is run in stage mode. The property stores the path to the location where the Advanced UI or Suite/Advanced UI installation is staged—or uncompressed and copied to a specified location.
ISSelectedLanguage	This property stores a string that contains the decimal language identifier of the language that the installation is currently displaying. Changing this property at run time, such as in the InstallationLanguage wizard page, changes the current language of the Advanced UI or Suite/Advanced UI installation. You can use this property to set the language of the packages in your Advanced UI or Suite/Advanced UI installation.
ISSilentInstall	If the Advanced UI or Suite/Advanced UI installation is running silently (without a user interface), this property is set to True.
ISUpdateAvailable	This property indicates whether an update for the currently running Advanced UI or Suite/Advanced UI installation is available for download. If an update is available, this property is set to 1. To learn more, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.
ISUpdateVersion	This property indicates the version of the Advanced UI or Suite/Advanced UI installation that is available for download. To learn more, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.

Advanced UI and Suite/Advanced UI Folder Properties

The following properties define where files are stored or installed on the target system.

Table 9-18 • Advanced UI and Suite/Advanced UI Folder Properties

Property	Description
ISLogDir	If logging is enabled and configured for one or more of the packages in the Advanced UI or Suite/Advanced UI installation, and if an end user launches the Advanced UI or Suite/Advanced UI installation from the command line with the `/log` parameter, this property stores path to the directory that contains the package log files. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.
ISPREREQDIR	This property stores the path to an Advanced UI or Suite/Advanced UI package's files when the package is staged, or copied to the target system for use temporarily during run time.
SETUPEXEDIR	This property stores the path to the Advanced UI or Suite/Advanced UI installation's **Setup.exe** file. For example, if the path to **Setup.exe** is **C:\MySetups\MyApp\Setup.exe**, the value of *SETUPEXEDIR* is **C:\MySetups\MyApp**.
SETUPEXENAME	This property identifies the name of the setup launcher file that was created when the project was built. The installation updates the value of this property at run time if the setup launcher file was renamed. The following path identifies the full path to this file: **[SETUPEXEDIR]\[SETUPEXENAME]**

SETUPSUPPORTDIR	This property stores the path to the folder that contains the support folders and files that are used temporarily by the Advanced UI or Suite/Advanced UI installation at run time and then removed. This location is sometimes referred to as the *default staging path*. To refer to a file that is in the Language Independent node in the Support Files view, use the following syntax:
	
	`[SETUPSUPPORTDIR]\MyLanguageIndependentFile.txt`
	
	To refer to a file that is in one of the language-specific nodes in the Support Files view, include the language identifier in the path. For example, the language identifier for English is 1033:
	
	`[SETUPSUPPORTDIR]\1033\MyEnglishFile.txt`

TempFolder	This property stores the path to the current temporary folder in the user context in which the Advanced UI or Suite/Advanced UI installation is running.

System Folder Properties

The following properties define paths to various folders on end users' systems.

Table 9-19 • System Folder Properties

Property	Description
AdminToolsFolder	The value of this property is the full path to the end user's folder that contains administrative tools.
AppDataFolder	The value of this property is the full path to the end user's Application Data folder.
CommonAppDataFolder	The value of this property is the full path to the All Users Application Data folder.
CommonDocuments	The value of this property is the full path to the folder that contains documents that are common to all users.
CommonFilesFolder	The value of this property is the full path to the 32-bit Common Files folder.
CommonFiles64Folder	The value of this property is the full path to the 64-bit Common Files folder on 64-bit systems. This property is undefined on 32-bit systems.
LocalAppDataFolder	The value of this property is the full path to the folder that contains data files for local (non-roaming) applications.
PersonalFolder	The value of this property is the full path to the end user's Personal folder.
ProgramFilesFolder	The value of this property is the full path to the end user's Program Files folder. On 64-bit systems, this property refers to the 32-bit location.
ProgramFiles64Folder	The value of this property is the full path to the end user's 64-bit Program Files folder. This property is undefined on 32-bit systems.
SystemFolder	The value of this property is the full path to the 32-bit system folder.
System64Folder	The value of this property is the full path to the 64-bit native system folder on 64-bit systems. This property is undefined on 32-bit systems.
WindowsFolder	The value of this property is the full path to the end user's Windows folder.

Creating Properties in Windows Installer–Based Projects

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Project-specific differences are noted where applicable.

For instructions on creating properties in Advanced UI and Suite Advanced UI projects, see Creating Properties in Advanced UI and Suite/Advanced UI Projects.

You can create your own project-wide properties through the Property Manager. These properties allow you to set a value in one place and use it throughout your project.



Task ***To create a property:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Click the **New Property** button. InstallShield adds a new row at the bottom of the view.
3. In the **Name** column, enter a name for the new property.
4. In the **Value** column, enter a value for the property.
5. In the **Comments** column, optionally enter comments about the property.



Tip *• Use the following guidelines when entering values in the Property Manager view:*

- *To overwrite all of the text in a table cell, click the table cell and then type your new property name, value, or comments.*
- *To place the cursor at a particular place within a table cell, double-click that place. Then type your change.*

Tips for Creating Windows Installer Properties in Windows Installer–Based Projects

If you want to create a property that can be changed at the command line, use all uppercase letters in the property's name. For example, INSTALLDIR is a property that can be set or changed from the command line. For more information, see Overview of Windows Installer Properties.



Windows Logo *• InstallShield does not verify whether the information that you enter in the Property Manager view is valid. For example, if you change the value of the* ***ARPHELPLINK*** *property to* ***MyCompany*** *instead of* ***http://www.mycompany.com****, the link fails when an end user clicks it. InstallShield does not display an error message when you enter the data in the Property Manager or at build time. To check whether you have the correct information and syntax for a built-in Windows Installer property, see the Windows Installer Property Reference.*

Creating Properties in Advanced UI and Suite/Advanced UI Projects



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

You can create your own project-wide properties through the Property Manager. These properties allow you to set a value in one place and use it throughout your project.



Task ***To create a property:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Click the **New Property** button. InstallShield adds a new row at the bottom of the view.
3. In the **Name** column, enter a name for the new property.
4. In the **Value** column, enter a value for the property.
5. In the **Formatted** column, select or clear the check box, which lets you indicate whether the text that you entered in the **Value** column is formatted.
   - To leave the property value (including square brackets and the content within them) as is, clear the **Formatted** check box.
   - If the value in the Value column contains one or more formatted expressions with property names, environment variable references, or other special strings that you want to be resolved at run time, select the **Formatted** check box. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.



Tip *• Use the following guidelines when entering values in the Property Manager view:*

- *To overwrite all of the text in a table cell, click the table cell and then type your new property name, value, or comments.*
- *To place the cursor at a particular place within a table cell, double-click that place. Then type your change.*

Changing an Existing Property



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

Task ***To change an existing property in your project:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Do one of the following:
   - To overwrite all of the text in a table cell, click the table cell and then type your new property name, value, or comments.
   - To place the cursor at a particular place within a table cell, double-click that place. Then type your change.

Creating a Localizable Property

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

If you want a property that can have different values based on the language that your installation uses, you can create a localizable property.

Task ***To create a localizable property:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Click the arrow next to the **New Property** button, and then click **Localizable Property**. InstallShield adds a new row at the bottom of the view. A new string identifier is listed in the Value column for that property.
3. In the **Name** column, enter a name for the new property.

Tip *• You can use the String Editor view to set a different property value for each language that your project supports.*

To learn how string identifiers and values are used in projects, see Using String Entries in InstallShield.

Making an Existing Property Localizable

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

If you want a property that can have different values based on the language that your installation uses, you can make that property localizable.

Task ***To make a property localizable:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Select one or more properties that you want to localize.

   To select multiple consecutive properties, select the first property, press and hold SHIFT, and select the last property. To select multiple nonconsecutive properties, select the first property, press and hold CTRL, and select each additional property.
3. Click the **Make Selected Properties Localizable** button.

InstallShield adds a new string identifier in the Value column for that property.

Tip *• You can use the String Editor view to set a different property value for each language that your project supports.*

To learn how string identifiers and values are used in projects, see Using String Entries in InstallShield.

Preventing a Property Value from Being Written in Windows Installer Log Files

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

By default, Windows Installer writes the final value of every Windows Installer property contained in your installation to a log file generated by launching MsiExec with the `/L` argument. Starting with Windows Installer version 2.0, you can prevent certain properties (such as those containing passwords) from being written in the log file.

Task ***To prevent a property from being written in the log file:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. In the **Name** column, find the **MsiHiddenProperties** property.

   If this property is not listed, click the **New Property** button to create this property, and in the **Name** column, enter **MsiHiddenProperties**.
3. In the **Value** column, enter the name of the property that you want to be hidden. To list more than one property, separate each with a semicolon (;).

For more information about this property, see MsiHiddenProperties in the Windows Installer Help Library.

Preventing a Property Value from Being Written in Advanced UI and Suite/Advanced UI Debug Log Files

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

If you launch an Advanced UI or Suite/Advanced UI **Setup.exe** file with the /debuglog parameter, the Suite engine generates a debuglog file. By default, the debug log file includes the values of changed Advanced UI or Suite/Advanced UI properties.

In some cases, you may want to prevent the Suite engine from writing the values of specific properties to the debug log file. For example, you may want to prevent properties that contain passwords and other sensitive information from being logged.

Task ***To prevent a property from being written in the log file:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. In the **Name** column, find the **ISHiddenProperties** property.

   If this property is not listed, click the **New Property** button to create this property, and in the **Name** column, enter **ISHiddenProperties**.
3. In the **Value** column, enter the name of the property whose value you do not want to be logged; ensure that you use the same capitalization. To list more than one property, separate each with a semicolon (;).

Note that **ISHiddenProperties** is useful for prevention of logging for values in the following scenarios:

- The value of the property changes.

- The end user sets the value of the property through command line when launching the Advanced UI or Suite/Advanced UI **Setup.exe** file.
- The property is configured through a command line that the Advanced UI or Suite/Advanced UI installation passes to a package. This is configurable in the Packages view, on the Common tab, in the Operation area.

If the property is logged through ISuiteExtension::LogInfo, the Suite engine cannot prevent the logging. Therefore, any code that you create to write a property value to the log file should read **ISHiddenProperties** to see whether the property value should be logged.

Specifying that a Public Property Should Be a Restricted Public Property



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Restricted public properties allow network administrators to define public properties that can be changed only by a system administrator or by someone who has elevated privileges. This way, the administrator can change settings quickly without having to worry that unauthorized users on the network may tamper with the installation.

Windows Installer considers a number of public properties to be restricted public properties. For the full list of restricted public properties, see Restricted Public Properties in the Windows Installer Help Library.

To include additional public properties, add them to the **SecureCustomProperties** property.

If you perform tasks such as the following ones, InstallShield automatically adds the applicable property to the **SecureCustomProperties** property:

- When you use a public property in a system search, InstallShield adds the public property to the **SecureCustomProperties** property.
- When you add or import a dialog to a Basic MSI or Merge Module project, and the dialog contains a control that is set through a property, InstallShield adds that property to the **SecureCustomProperties** property.
- When you add a major upgrade item to your project, InstallShield adds the value of the Detect Property setting on the Advanced tab for that upgrade item to the **SecureCustomProperties** property.

This enables the custom public properties to be set in the user interface sequence and then be passed to the execute sequence.



Task ***To manually specify that a custom public property should be a restricted property:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. In the **Name** column, find the **SecureCustomProperties** property.

If this property is not listed, click the **New Property** button to create this property, and in the **Name** column, enter **SecureCustomProperties**.

3. In the **Value** column, enter the public properties that you would like to restrict. Separate multiple entries with a semicolon (;).

Getting or Setting Windows Installer Properties in InstallScript



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

The **MsiGetProperty** and **MsiSetProperty** functions get and set Windows Installer properties. For an example, see Getting and Setting Properties.

You must include the statement `#include "iswi.h"` or `#include "ifx.h"` in order for the Windows Installer APIs to be available to your script.

Removing a Value from a Property



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

Advanced UI and Suite/Advanced UI projects do not have support for localizable properties.

You can clear a property's value without deleting the property from the Property Manager. This functionality works for localizable and non-localizable properties. For example, you might want to clear a localizable property such as **ARPCOMMENTS**.

Task ***To clear a property value:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Select one or more properties whose values you want to clear.

   To select multiple consecutive properties, select the first property, press and hold SHIFT, and select the last property. To select multiple nonconsecutive properties, select the first property, press and hold CTRL, and select each additional property.
3. Click the **Clear Selected Properties** button.

Note • *Clearing a property makes the property non-localizable because it does not have a string identifier.*

Deleting a Property

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

If you no longer need a property in your project, you can delete it from the Property Manager view.

Task ***To delete a property from your project:***

1. In the View List under **Behavior and Logic**, click **Property Manager**.
2. Select one or more properties that you want to delete.

   To select multiple consecutive properties, select the first property, press and hold SHIFT, and select the last property. To select multiple nonconsecutive properties, select the first property, press and hold CTRL, and select each additional property.
3. Click the **Delete Selected Properties** button.

InstallShield displays a message box that lets you specify whether you want to delete just the selected properties, or the selected properties plus any associated string entries.

Directly Editing .msi and .msm Databases

The MSI Database and MSM Database project types let you edit Windows Installer installation databases and merge module databases directly, rather than working through an intermediate project format (.ism file). These project types extend the Direct Editor functionality to include different views that are supported in standard installation projects. The views that are supported in these project types are intended to function as they do for an .ism project.



***Project** • There is no string entry or path variable support in an MSI Database or MSM Database project.*

You can directly edit an existing .msi file or .msm file, or directly create a new database by opening a new MSI Database or MSM Database project.

Opening Windows Installer Packages

InstallShield provides the option to either edit .msi packages directly in InstallShield by using direct edit mode or convert it to an InstallShield project (.ism), and then edit it as a Basic MSI project in InstallShield.



***Note** • Some functionality is limited when editing an .msi package in direct edit mode.*



Task ***To open an .msi package in direct edit mode of InstallShield:***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. In the **Files of Type** list, select **Windows Installer Packages (*.msi)**.
3. Select the .msi package that you want to open.
4. In the **Open as** box, ensure that **Auto** is selected.
5. Click **Open**.



***Note** • By default, InstallShield opens an .msi package in direct edit mode.*

The Open MSI/MSM Wizard can convert an existing Windows installer (.msi) package to an InstallShield project (.ism) file, and open the new project to modify in InstallShield.



***Note** • You can also open patch creation project files (.pcp files) in InstallShield and edit them in the Direct Editor.*

Editing .msi and .msm Databases in Direct Edit Mode

If you want to directly edit a Windows Installer package (.msi file) or merge module (.msm file), you can open it in InstallShield in direct edit mode and then make the required changes.

Task ***To open an .msi or .msm file:***

1. On the **File** menu, click **Open**.
2. In the **Files of Type** list, select **Windows Installer Packages (*.msi)** or **Windows Installer Modules (*.msm)**.
3. Browse to the Windows Installer package or merge module that you want to open.
4. Click **Open**.

The selected package or merge module opens in a limited view of the InstallShield interface. Any changes made in the direct edit mode are made directly to the .msi or .msm file when you save your changes.

Adding Files in Direct Edit Mode

When adding files in direct edit mode, you have the option to add files to the media in an uncompressed, a compressed, or a compressed and streamed state into the MSI format (MSI package or MSI database). These options appear in the Select Media Location dialog box when you drag and drop new files in the Files and Folders view in direct edit mode. You also have the option to extract COM information in that dialog box. However, this option is enabled only when dragging files to a folder or when adding a single file to a component that does not already have any existing files. As a result, the COM file is the only file in that particular component.

Deleting files removes the entry from the **File** table. Files on the media are not deleted, and files that have yet to be added to the media are not added until you save.

Note *• In direct edit mode, file and media tables are updated during the save process.*

When a file is added, a record is created with a default sequence value. No **Media** table entry is made at this time. When the media or project is saved, the sequence for the newly added files is updated to correctly reflect the media, and media entries are created in the **Media** table.

Adding Merge Modules in Direct Edit Mode

When you add a configurable merge module or object in the Redistributables view to an .msi database in direct edit mode, the Merge Module Properties dialog box opens. You can review or configure the merge module's properties through that dialog box. However, you can do this only when the merge module is initially merged with your installation database. That is when this dialog box is first opened. You should also note that a merge occurs as soon as you select a merge module in direct edit mode.

Note *• You cannot configure or set the destination of dependent merge modules.*

You cannot un-merge modules that are not found in the merge module catalog. For example, a file is not found in the Modules\i386 folder. In that case, the check box next to the respective merge module is selected but disabled.

Once you add, configure, and merge the merge modules, any files included in the merged modules will be copied to the source media upon saving the database in InstallShield.

10

Integrating InstallShield with External Applications

An important aspect of InstallShield is how it coexists and integrates with other software development tools such as Visual Studio as well as source code control software that complies with the Microsoft Source Control Interface.

Consult the help topics in this section for specific details on the extent of external application support in InstallShield.

Using Source Code Control

InstallShield enables you to manage different versions of your project file in source code control software. InstallShield is capable of interacting with any source control system that complies with the Microsoft Source Control Interface and uses the default program on the development system. If none is installed, the source control options are not available.



***Tip** • InstallShield also includes support for integrating with Team Foundation source control. To learn more, see Integrating with Microsoft Visual Studio Team Foundation Server.*

When you add an installation project to source control, InstallShield automatically converts the project file format to text (XML format) and maintains it in the source control system by default. If you choose not to check in your file in XML format, then the file will be checked in as a binary file.



***Note** • When a project file format is converted to either XML or binary format, the project file extension remains .ism.*

Script File Support

InstallShield also supports the addition of any InstallScript files and SQL scripts, regardless of whether you insert, import, or add new files. It also supports all path variable types that you define in the Path Variables view to define the location of these script files. However, it is important to be aware that a corresponding folder structure exists in your source code database so that your source code control software can resolve paths.

Using Source Code Control Integration

Task ***A typical scenario for using InstallShield's source code control integration is outlined below:***

1. Create an InstallShield installation project.
2. Add the setup project to your source control program.
3. Check the project out of source control.
4. Edit the project.
5. Save the project.
6. Check the project back in to source control.

Tip *• You can streamline this process by having InstallShield automatically add new projects to source control or check out edited projects. For more information, see the Source Control tab of the Options dialog box; InstallShield displays this dialog box when you click Options on the Tools menu.*

Adding Projects to Source Control

InstallShield lets you add an InstallShield project to your source code control system after you have created your InstallShield project.

Tip *• You can omit steps 1 and 2 if you selected the* ***Add new projects to source control*** *check box on the Source Control tab of the Options dialog box; InstallShield displays this dialog box when you click Options on the Tools menu. As soon as you create the project, InstallShield adds it to your source code control system.*

Task ***To add a project to your source control system after you have created the project:***

1. Open the project in InstallShield.
2. On the **Project** menu, point to **Source Control**, and click **Add to Source Control**. The Add to Source Control dialog box opens.
3. Add comments (if applicable) and indicate whether you want to keep the project checked out.
4. Click **OK**. A log-on dialog box for your organization's source control system opens.
5. Enter your log-on information.
6. Browse to the location in your source control database where you want to add the project file.

Note *• Linked files are not added to source control.*

When you add a file to source control, most programs make the local copy read only. If this is the case with your source control software, you need to check out the project file before making any changes.

Checking Projects out of Source Control

InstallShield lets you check out an InstallShield project from your source code control system.



Tip *• You can omit steps 1 and 2 if you selected the* ***Check out the project when edited*** *check box on the Source Control tab of the Options dialog box; InstallShield displays this dialog box when you click Options on the Tools menu. As soon as you make a change to the project (anything that enables the Save button on the toolbar), InstallShield adds it to your source code control system.*



Task ***To check a project out of your source code control program:***

1. Open the project in InstallShield.
2. On the **Project** menu, point to **Source Control**, and click **Check Out**.
3. If you selected the **Use dialog for checkout** check box on the **Source Control** tab of the **Options** dialog box, you are prompted for your comments in the **Check Out** dialog box.
4. Enter comments as appropriate, and then click **OK**.

Your project is now checked out of your source control program, and you can edit the local copy before checking it in.



Note *• The project is not automatically checked in to source control when you close the project in InstallShield. Check your project in (from within InstallShield) before closing it.*

Checking Projects in to Source Control



Task ***To check a project in to your source control program from within InstallShield:***

1. Open the project in InstallShield.
2. On the **Project** menu, point to **Source Control**, and click **Check In**.
3. The **Check In** dialog box prompts you for options for checking in the file. Click **OK**.

Your project is now checked into your source control program, which typically means that your project file is set to read only until you check it out of source control.



Note *• The project is not automatically checked in to source control when you close the project in InstallShield. Check your project in (from within InstallShield) before closing it.*

Unlinking Projects from Source Control



Task ***To break the link between the project on your local machine and the one in your source control application:***

On the **Project** menu, select **Source Control**, and point to **Unlink from Source Control**.

Integrating with Microsoft Visual Studio

With InstallShield, you can create your installation projects directly in Microsoft Visual Studio. InstallShield enables you to create, modify, or build your installation from within Microsoft Visual Studio.

Integration Features

InstallShield is fully integrated within the Visual Studio shell. Some of the unique features of the integration include:

- All InstallShield navigation is presented within the Solution Explorer.
- Each InstallShield view is presented in a separate window so no scrolling is necessary and side-by-side viewing options are available.
- You can run InstallShield external to Visual Studio.
- InstallShield dynamically links your installation project to other Visual Studio project outputs, automatically updating your installation with your latest source files every time your product is built.
- Items identified by the InstallShield Debugger can be moved directly into the Visual Studio task list.

Integration Benefits

Some additional benefits of using InstallShield's integrated installation authoring solution are:

- You can create and customize your installation without leaving the Visual Studio user interface, enabling use of familiar navigation and layout options.
- Your installation is automatically updated with your latest source files every time your solution is built, always staying current.
- The installation reflects the Build Configuration for the solution—for example, Debug, Release, automatically included source files from the proper build directory.
- .NET properties and dependencies can be scanned and included in an installation automatically.



Note *• If you want to create, edit, and build your InstallShield projects directly within Visual Studio, you must use Visual Studio 2005 or later. InstallShield cannot be integrated with Visual Studio 2003 or earlier.*

Visual Studio can be integrated with only one version of InstallShield at a time. The last version of InstallShield that is installed or repaired on a system is the one that is used for Visual Studio integration.

Creating InstallShield Projects in Microsoft Visual Studio

InstallShield is integrated with Microsoft Visual Studio. From within the Visual Studio workspace, you can create InstallShield installations for solutions.

Task ***To create an InstallShield project from within Microsoft Visual Studio:***

1. On the **File** menu, point to **New** and click **Project**. The **New Project** dialog box opens.
2. *Beginning with Visual Studio 2010:* In the **Installed Templates** box, click **InstallShield Projects**. Then select the appropriate project type.

   For earlier versions of Visual Studio: In the **Project Types** box, click **InstallShield Projects**. Then in the **Templates** box, select the appropriate project type.
3. Configure the settings for the name and project location as needed.
4. Click **OK**.

Opening InstallShield Projects in Microsoft Visual Studio

InstallShield is integrated with Microsoft Visual Studio. From within the Visual Studio workspace, you can open InstallShield installations for solutions.

Task ***To open an InstallShield project from within Microsoft Visual Studio:***

1. On the **File** menu, point to **Open**, and choose **Project** to display the **Open Project** dialog.
2. Browse to the InstallShield file you want to open, and select it.
3. Click **Open**.

Note *• From within Microsoft Visual Studio, you can open only .ism files created with InstallShield X or later, InstallShield DevStudio, InstallShield Developer, InstallShield—Windows Installer Edition, or InstallShield Express, version 3.x or later. Files created in InstallShield Professional or pre-3.x versions of InstallShield Express cannot be opened.*

Using the VSSolutionFolder Path Variable with Visual Studio Solutions

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*

- *InstallScript MSI*
- *Merge Module*
- *Suite/Advanced UI*

A predefined path variable called VSSolutionFolder is available in projects to reference a higher-level base directory. This support enables you to have in your InstallShield projects static links to files in sibling projects that are within the Visual Studio solution folder; if you work on the projects on a different machine, the static links that use the VSSolutionFolder path variable can reference the correct paths for the files in sibling projects.

The VSSolutionFolder path variable is defined automatically whenever an InstallShield project is opened from within a Visual Studio solution. It is also defined automatically if you are using MSBuild to build a solution that contains an InstallShield project. However, in other scenarios, when the InstallShield project is opened without the Visual Studio solution, VSSolutionFolder cannot be defined automatically. For example, if you open the InstallShield project in InstallShield directly, without having Visual Studio open, VSSolutionFolder is not defined. Similarly, if you use the command-line tool **IsCmdBld.exe**, or if you use MSBuild with an .isproj file, VSSolutionFolder is not defined. If you are using **IsCmdBld.exe** to build a release in InstallShield project, use the `-L` command-line parameter to set the value of VSSolutionFolder. If you are using MSBuild, use the PathVariables parameter to set the value of VSSolutionFolder. This parameter is exposed as the ItemGroup InstallShieldPathVariableOverrides when the default targets file is used.

If you include in your InstallShield project a source file whose path includes the VSSolutionFolder path variable and build it in an environment that does not support the VSSolutionFolder path variable, build errors such as the following ones may occur:

- -6103: Could not find file <VSSolutionFolder>\MyFile.exe
- -6271: File <VSSolutionFolder>\MyFile.exe not found. An error occurred building the MsiFileHash table record for this file. Verify that the file exists in the specified location.

Adding References to Visual Studio Solutions

Use the Files and Folders view to add Visual Studio references to your installation project.



Task ***To add a reference:***

1. In the **Solution Explorer** under **Application Data**, double-click **Files and Folders**.
2. The **Visual Studio Solution** node in the **Source computer's folders pane** contains sub-nodes for all projects contained in the current solution. Click a project to display the output groups for the project. The output groups are displayed in the **Source computer's files pane**.
3. To add a reference to an output to your installation project, drag and drop the output to the target folder in the **Destination computer's folders** pane.



Note • *To view a reference for a project, right-click the project and select Resolve Project Output. The Outputs dialog is displayed to show what files the output group resolves to.*

Using the Toolbox to Add Dialog Controls

To add dialog controls when your setup project is open in Microsoft Visual Studio, use the Toolbox.

Accessing the Toolbox



Task ***To access the InstallShield Dialogs Toolbox:***

1. Do one of the following:
   - On the **View** menu, click **Toolbox**.
   - Press **CTRL+ALT+X**.
2. Click the **InstallShield Dialogs** tab, if it is not already expanded.

Adding Controls



Task ***To add controls to a dialog:***

1. Click on the type of control you want to add to the dialog.
2. Click on the dialog, hold the mouse button down, and drag the mouse to draw the dimensions of the control. You can adjust the control's size by setting the height and width properties.

   As soon as you release the mouse button, the cursor changes to the Select Tool, which you can use to resize the new control.
3. Set the control's properties. For more information, see Editing Dialog Layout in Basic MSI Projects or Editing the Layout of a Dialog in an InstallScript or InstallScript MSI Project.

Using Sticky-State Mode

The toolbox control also has a *sticky-state* mode that allows you to create additional controls of the same type without returning to the Toolbox. To enable sticky-state mode, press the CTRL key while you drag the cursor from the toolbox to the dialog.

Using Auto Hide

The Dialog Toolbox has an auto-hide feature that allows you to hide the toolbox when it is not in use. To enable Auto Hide, click the Auto Hide button in the top-right corner of the toolbox. The toolbox is hidden on the side of the screen when you are not using it.

Adding InstallShield Toolbars or Commands to the Visual Studio Toolbar

You can add InstallShield toolbars and individual toolbar command buttons to the Microsoft Visual Studio workspace.

Adding InstallShield Toolbars

Task ***To add an InstallShield toolbar:***

1. Right-click anywhere in the toolbar to display the toolbar options.
2. Select a toolbar to add it to the top of the Visual Studio workspace. The available InstallShield toolbars are InstallShield, InstallScript, InstallShield Layout, and MSI Debugger.

Adding InstallShield Toolbar Command Buttons

Task ***To add an individual command button to the toolbar:***

1. Right-click anywhere in the toolbar to display the toolbar options.
2. Select **Customize** from the bottom of the list. The **Customize** dialog is displayed.
3. Click the **Commands** tab.
4. Select a category from the **Categories** list to display the commands available within that particular category.
5. From the **Commands** list, click a command button and drag it to the toolbar.

Note • *InstallShield Layout and MSI Debugger toolbar commands are not available as individual toolbar buttons.*

Building Releases in Microsoft Visual Studio

Project • *Some of the following information is project specific, as noted where applicable.*

Building a release of an InstallShield project in Visual Studio is different than building a release in InstallShield. When you are building in Visual Studio, you have the option to either build your entire solution, including the installation project, or build only the installation project.

When you create an InstallShield installation project in Visual Studio, the project has two releases by default—Debug and Release. You can use these default releases, or you can create new release. Before you can build a release, the release must be associated with a solution configuration. The Configuration Manager in Visual Studio is where you associate a release with a solution configuration.

Project • *In Basic MSI, InstallScript MSI, and Merge Module projects, every release is a member of a product configuration. A product configuration provides a means for grouping together releases that share similar settings, such as the product name, product code, and package code. When you build a release for any of these project types, the default product configuration is always used. If you want to build releases in a different product configuration, make that product configuration the default.*

Selecting the Default Product Configuration (Basic MSI, InstallScript MSI, and Merge Module Projects Only)

If you want to build a release in a Basic MSI, InstallScript MSI, or Merge Module project, it must be under the default or *active* product configuration. This is indicated by a red check box icon in the Releases view:



Task ***To make a product configuration the default configuration in your InstallShield project:***

1. In the **Solution Explorer** under **Media**, double-click **Releases**.
2. In the **Releases** explorer, right-click the product configuration that contains the release that you want to build, and then click **Default Product Configuration**.

Tip • *To create a new product configuration, right-click the* ***Releases*** *explorer and then click* ***New Product Configuration****. Once you have created it, you can make it the default product configuration.*

Building a Release

Visual Studio enables you to build your entire solution, including the installation project, or only the installation project.

Task ***To build a release from within Visual Studio:***

1. Use the Configuration Manager in Visual Studio to map a project configuration to the appropriate solution configuration.
   a. On the **Build** menu, click **Configuration Manager**.

      As an alternative: On the **Standard** toolbar, in the **Solution Configurations** list, select **Configuration Manager**.

   b. In the **Active solution configuration** list, select a configuration.

   c. In the **Project contexts** box, map the project configuration to the appropriate release in the **Configuration** column.

      Note • *If the Build check box is cleared and you build the solution, the deselected project configuration is not built.*

   d. Click the **Close** button.

2. On the **Build** menu, select the appropriate command:

Table 10-1 • Build Menu Commands

Command	Description
Build Solution	Build the entire solution, according to what is mapped in the Configuration Manager for each project that is included in the solution.
Build *Installation Project Name*	Build only the installation project, according to the release that is specified in the Configuration column of the Configuration Manager. This option is available if your installation project is selected in the Solution Explorer.



Tip • *To build the entire solution, you can also press CTRL+SHIFT+B.*

Adding .NET Assemblies to Installation Projects

InstallShield allows you to add a .NET assembly to your installation project by adding the .NET assembly file to a component.



Task ***The recommended way to add a .NET assembly to your installation project is as follows:***

1. In the **Setup Design** view, create a component to hold the .NET assembly.
2. (Windows Installer based projects only) Make the file the key file of a component by right-clicking on the file and selecting **Set Key File** from the context menu.
3. Set the project to scan for dependencies by doing one of the following:
   - Set the **.NET Scan at Build component** property to Dependencies and Properties. In this case, the .NET assembly's Manifest, Application, and related entries are not populated in your project file. Instead, they are added to the .msi file at build time.
   - Run the Static Scanning Wizard. In this case, the .NET assembly's values are added during the scan. You can edit these values in the **Assembly** node under the component's **Advanced Settings**.

Adding Project Output from a Web Service or Web Application Project

InstallShield provides enhanced Web services support. If you add a project output (any project output) from a Web Service or Web Application project, InstallShield will prompt you to add the project as a Web Service. If you choose No, the project output that you have selected is added normally. If you select Yes, InstallShield performs the following:

1. Creates a Destination Folder called IISROOTFOLDER.
2. Deploys the following Visual Studio Project Outputs into the IISROOTFOLDER:
   [Content Files] goes to the [IISROOTFOLDER]{VSIPProjectName}
   [Primary Output] goes to the [IISROOTFOLDER]{VSIPProjectName}\bin"
3. Creates an IISVirtualDirectory with a target of [IISROOTFOLDER]{VSIPProjectName}.

Integrating with Microsoft Visual Studio Team Foundation Server

Microsoft Visual Studio Team Foundation Server (TFS) is a set of tools and technologies that enable a team to collaborate and coordinate the tasks for developing a product. InstallShield has support for integrating with Team Foundation Server. Some highlights of this integration are:

- **Source control**—Use the Source Control Explorer to integrate your InstallShield project with Team Foundation version control and manage changes to your InstallShield projects and your Visual Studio solutions.
- **Automated builds**—Use Team Foundation Build to compile, test, and deploy your InstallShield projects and your Visual Studio solutions on a regular basis. Your installation is automatically updated with your latest source files every time your solution is built, always staying current.
- **Project management**—Track work items such as bugs, tasks, and project documentation for your InstallShield projects and your Visual Studio solutions. The project status is available to your entire team from within Team System Web Access, and from within Team Explorer.

Integration Requirements

To integrate InstallShield with Team Foundation Server, install InstallShield on each machine that you want to be able to create, update, or build InstallShield projects. Thus, InstallShield should be installed on each machine on which you want to create and update InstallShield projects. It should also be installed on a machine that is designated as a build agent for InstallShield projects that are stored in Team Foundation Server. For InstallShield licensing details, refer to the InstallShield End-User License Agreement (EULA).

Note • *The Standalone Build is a build engine that enables you to build InstallShield projects without installing the full version of InstallShield on a build machine. If you have the Standalone Build, you can install it on a machine that is designated as a build agent for Team Foundation Server.*

In order for a build agent to build some types of projects and solutions, you may need to install additional software on the build machine. For example, to build a C++ project, which requires the C++ compiler and possibly other dependencies, install Visual Studio on the build machine.

If you are using multiple build agents to build your Team Foundation Server projects, you may want to assign a particular build tag to any agents that are on machines that have InstallShield; you could also apply that special build tag to each build definition that is created for an InstallShield project. That way, only build machines that have InstallShield installed would be used to build InstallShield installations. For more information about creating build tags and assigning them to agents and build definitions, see the Visual Studio Team Foundation Server documentation.

If you are queuing a build on a 64-bit build machine, ensure that you have configured the build definition for your InstallShield project so that the 32-bit version of MSBuild is used to load the **InstallShield.Tasks.dll** file (which is a 32-bit file); otherwise, you will encounter a build error informing you that the **InstallShield.Tasks.dll** file could not be loaded. To select the 32-bit version of MSBuild, click the Process tab of your build definition in Team Explorer. Then, under the Advanced node, find the MSBuild Platform setting, and select x86. Note that if you are using a 32-bit build machine, you can select either Auto or x86 for the MSBuild Platform setting.

If you install Team Explorer on the same machine that has InstallShield and Visual Studio, you can use Team Explorer from within your InstallShield projects that are open in Visual Studio. This enables you to perform tasks such as the following:

- Use Source Control Explorer when you are working on your InstallShield projects.
- Configure builds for your InstallShield projects and Visual Studio solutions.
- Queue new builds.

Note that when you queue a build for a solution that includes an InstallShield project, the installation that is built is copied to an Install subfolder within the drop folder. When the InstallShield build detects that it is running under Team Foundation Build, it copies the installation to the final output location for the solution (OutDir)—namely, the binaries directory, which in turn is copied to the drop folder at the end of the Team Foundation Build process.

If you want to use Team Foundation source control when you are working on your InstallShield projects directly from within InstallShield (without Visual Studio integration), you can do so. To use this functionality, you need to install the Team Foundation Server MSSCCI Provider, which you can obtain from the Visual Studio Gallery on the MSDN Web site. Once the Team Foundation Server MSSCCI Provider is installed, you can add InstallShield projects to Team Foundation source control, check out InstallShield projects from Team Foundation source control, and check in projects to Team Foundation source control—all from within InstallShield. To learn more about source control integration within InstallShield, see Using Source Code Control.

Adding InstallShield Projects to Team Explorer

If you have Team Explorer installed on the same machine that has InstallShield and Visual Studio, you can add your InstallShield projects (.ism and .isproj files, as well as any corresponding InstallScript files and SQL scripts) to the Team Foundation Server through Team Explorer.

If you want to add an InstallShield project to Source Control Explorer when you create the project from within Visual Studio, select the **Add to source control** check box on the New Project dialog box. Note that if you do this, you may later need to add InstallScript files and SQL script files to source control using the same method that you use for adding other files.

To add an existing InstallShield project to Source Control Explorer, add it using the same method that you use for adding other projects. (In the Solution Explorer, right-click your InstallShield project and then click Add Solution to Source Control.)

11

Automating Installation Development and Build Processes

An automation interface is loosely defined as a programming interface that can be accessed by any programming language that supports COM—most importantly, usability from the late binding scripting languages like VBScript and JScript. The InstallShield automation interface is an automation interface to the InstallShield project file.

InstallShield enables you to automate development and build processes through the automation interface without having to directly open the InstallShield interface and make changes in different views. By calling methods, setting properties, accessing collections, and so on, through the automation interface, you can open a project and modify its features and component data in many of the same ways that you would in the InstallShield interface. In addition, you can build a release using the Build method of the InstallShield automation interface.

Read more about the automation interface as well as other aspects of automating build processes in this section of the documentation.

Linking Subfolders to Features Using VBScript

The VBScript example simulates dynamically linking to subfolders. By editing .ini files, you tell the script which folders and subfolders to link to, how to create the components for the files, and with which features to associate the new components.

The following are requirements for using the VBScript example:

- You must have Windows Scripting Host to run a .vbs file.
- InstallShield must be installed on the system before you can use the automation interface and .ini file reader.
- The script expects two command-line arguments:
  - The fully qualified path to an existing setup project (.ism file).
  - The fully qualified path to an .ini file (Links.ini in this example) that describes each feature that you want a given folder's files and subfolders added to. Furthermore, each feature can point to a different .ini file (Template.ini) that describes the properties of the components that will be created for the files in each folder and subfolder.

The source files for this example are found in the following directory:

*InstallShield Program Files Folder***Samples\WindowsInstaller\Automation Interface\Add Files and Components**

Each of the files is described below.

LinkToFeature.vbs

The script in the **LinkToFeature.vbs** file is designed to read the associated Links.ini file and add the files from the specified folders and subfolders to the specified features. It organizes them into components based on the model you construct in Template.ini. To start the script, pass it the path to your setup project and the path to Links.ini, respectively, at the command line:

```
LinkToFeature.vbs "C:\MySetups\Malaprop.ism" "C:\Automation Interface\Links.ini"
```

You can examine some of the relevant sections of LinkToFeature.vbs to see exactly how it works. After opening the .ism file specified at the command line, the script creates a collection of every feature in the project starting at line 53. This collection is later used to make sure that the features listed in Links.ini are present in the project.

```
' The scripting dictionary is defined in Scrrun.dll
Dim pFeatCol
Set pFeatCol = CreateObject("Scripting.Dictionary")

' Create a collection of all of the feature names in the project
For Each ISWIFeature In ISWIProject.ISWIFeatures
    pFeatCol.Add ISWIFeature, ISWIFeature.Name
    DoSubFeature ISWIFeature, pFeatCol
Next
```

Next, **LinkToFeature.vbs** opens Links.ini using the InstallShield IniReader. As the comment on line 70 explains, you can use this utility if it serves your needs or otherwise substitute your own. Once Links.ini is open, the rest of the script is controlled by the For loop beginning at line 80 and ending at line 202:

```
For Each pLinksSection In LinksIniFile.Sections
```

The purpose of this loop is to read each section in Links.ini. Each section contains the name of a feature that you want to add the new components to and the folder where the files and subfolders can be found. After getting each key's value, there is a nested For loop, stretching from line 102 to 201, which loops through every feature in the collection pFeatCol and compares it to the name of the feature in Links.ini:

```
  For Each ISWIFeature In pFeatCol
        ' See if the feature in Links.ini matches one in the setup project.
        If (ISWIFeature.Name = linksFeature) Then
```

Since this loop is used for almost the remainder of the script, you can assume that everything that is done acts on the current feature. Next, it checks if the Action key in this Links.ini section is set to Delete and deletes all of the feature's components if so:

```
If (linksAction = "Delete") Then
    For Each ISWIComponent In ISWIFeature.ISWiComponents
        ISWIProject.DeleteComponent ISWIComponent
    Next
End If
```

At line 27, the script creates a collection of the specified folder and each subfolder:

```
Set pFolders = CreateObject("Scripting.Dictionary")
pFolders.Add folder, folder.Path
GetSubFolders pFolders, folder
```

From line 136 to 163, the script opens the **Template.ini** file that contains the template for all of the components that will be added to the current feature:

```
Set iniFile = CreateObject("IniReader.IniFile")
iniFile.Load (linksTemplateFile)
```

LinkToFeature.vbs does its real work in the For loop that begins on line 169 and ends on 199:

```
' Create a component for the root folder and all of its subfolders.
For Each folder In pFolders
```

Based on the description in Template.ini, it forms the component's name, adds it to the project, associates it with the current feature, sets the component's properties, and then adds the files in the current folder to the component.

Finally, the .vbs file saves and closes the setup project.

Links.ini

Each section of this .ini file contains the following keys, which LinkToFeature.vbs reads to determine which folders and subfolders to add to each feature in your project:

Table 11-1 • .ini File Keys in Sample File

Key	Description
[LinkN]	`;Unique section name`
Feature=FeatureName	`;Name of existing feature in the project`
Source=C:\Data Files\Executables	`;Path to the root folder`
TemplateFile=Template.ini	`;Path to the component template .ini file`
ComponentTemplate=Template1	`;Section name in Template.ini for this feature's components`
Action=Delete	`;Delete tells LinkToFeature.vbs to permanently delete all of this feature's components before recreating them. Leave this value blank if you do not want the components destroyed.`

Template.ini

Each section in Template.ini contains information for creating all of the components that will be associated with a feature. Each section in Links.ini must point to a specific component template file and section. The template file and sections are described below:

Table 11-2 • Template.ini File Sections

Section	Description
[TemplateName]	`;Unique section name that identifies a particular template`
Name=DefaultName	`;This value is used to build the component's name in the format DLS_DefaultName_`*N*

Table 11-2 • Template.ini File Sections (cont.)

Section	Description
Destination=[ProgramFilesFolder]	`;This is the root destination folder for the components. The names of the subfolders are then added to the root so that the files are installed to the correct relative folders.`
Languages=1033	`;List the decimal value of all of the component's languages, separated by a comma`
Condition=VersionNT	`;Author any condition you want to attach to each component`

Exporting and Importing String Entries Using the Automation Interface

InstallShield stores string entries in the **ISString** table of the main project file (.ism file). InstallShield lets you export the string entries of a language to a text file, which you can send out for translation. After the strings have been translated, you can import the text files into the InstallShield project file.

The String Editor view in InstallShield enables you to manually export and import string entries for a particular language. To automate this process, you can use the InstallShield automation interface.

One of the functions of the automation interface is to provide you with a mechanism for exporting string entries from and importing string entries into a project. What follows is an example of how to achieve this.

Running the Sample VBScript File

To run this sample, you must first copy the sample code shown below into a file called **ImportExportStrings.vbs**. Then use the command line to export or import the string entries.

Exporting a Language's String Entries

To export string entries using the code sample, type the following at the command line.

```
wscript.exe ImportExportStrings.vbs "C:\MyProject.ism" "C:\1033.txt" "1033" "E"
```

Importing a Language's String Entries

To import string entries using the code sample, type the following at the command line.

```
wscript.exe ImportExportStrings.vbs "C:\MyProject.ism" "C:\1033.txt" "1033" "I"
```

Sample ImportExportStrings.vbs Code

```
'////////////////////////////////////////////////////////////////////////
'BEGIN ImportExportStrings.vbs
'
'  This utility was designed to assist in the exporting and importing of string entries
'
'
'  File Name: ImportExportStrings.vbs
'
'  Description: Sample VBScript file exports or imports string entries in a text file
```

```
'
'  Usage:  To use the objects in this script, make sure you have the
'          following items in place on your system:
'          1. InstallShield must be installed so that
'             the end-user automation is available.
'          2. You must have Windows Scripting Host (Wscript.exe) installed.
'          3. The script expects the following command-line arguments,
'             in this order:
'             a. The fully qualified path to an existing .ism file.
'             b. The fully qualified path to a text file that contains the string entries
'             c. Decimal language identifier
'             d. Import or Export
'
'//////////////////////////////////////////////////////////////////////////


If Wscript.Arguments.Count < 1 Then
    Wscript.Echo "InstallShield Subfolder Utility" & _
        vbNewLine & "1st argument is the full path to the .ism file" & _
        vbNewLine & "2nd argument is the full path to the string text file" & _
        vbNewLine & "3rd argument is the decimal language identifier" & _
        vbNewLine & "4th argument is either E or I for Export or Import"
    Wscript.Quit 1
End If

' Create the end-user automation object
Dim ISWIProject
Set ISWIProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject"): CheckError

' Open the project specified at the command line
ISWIProject.OpenProject Wscript.Arguments(0): CheckError

if(Wscript.Arguments(3)="E")then
    StringsExport ISWIProject,Wscript.Arguments(1),Wscript.Arguments(2)
elseif(Wscript.Arguments(3)="I")then
    StringsImport ISWIProject,Wscript.Arguments(1),Wscript.Arguments(2)
end if

' Save and close the project
ISWIProject.SaveProject: CheckError
ISWIProject.CloseProject: CheckError

'//////////////////////////////////////////////////////////////////////////
' Export the language's string entries
Sub StringsExport(byref p,byval path, byval language)
    p.ExportStrings path, Date , language
    Wscript.Echo "Exported String entries for language " & language & " to " & path
End Sub

'//////////////////////////////////////////////////////////////////////////
' Import the string entries
Sub StringsImport(byref p,byval path, byval language)
    p.ImportStrings path, language
    Wscript.Echo "Imported string entries for language " & language & " from " & path
End Sub

'//////////////////////////////////////////////////////////////////////////
Sub CheckError()
    Dim message, errRec
```

```
    If Err = 0 Then Exit Sub
    message = Err.Source & " " & Hex(Err) & ": " & Err.Description
    Wscript.Echo message
    Wscript.Quit 2
End Sub

'END ImportExportStrings.vbs
```

Running Project Reports Using the Automation Interface

This VBScript example below accesses objects in a setup project to display a report of the total number of features, components, files, and various types of files. The output is displayed in a message box, but you could also write this information to a text file.

Requirements

The following requirements should be met:

- You must have Windows Scripting Host to run a .vbs file. (It would be very simple to modify this script to run in a Visual Basic program.)
- InstallShield must be installed on the system before you can use the automation interface.
- Before running the script, assign the fully qualified path of an existing setup project (.ism file) to the string variable sProj.

Example Script

```
Option Explicit

' Create automation object
Dim pProj
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")

' Open a setup project--make sure it is not already open in the IDE
Dim sProj
sProj = "C:\MySetups\TestProject.ism"
pProj.OpenProject sProj

' Start building string that lists features, components, and files
Dim sItems
sItems = "Project Report" & vbNewLine
sItems = sItems & "For project: " & sProj & vbNewLine
sItems = sItems & "Number of features: " & pProj.ISWiFeatures.Count & vbNewLine
sItems = sItems & "Number of components: " & pProj.ISWiComponents.Count & vbNewLine

' Declare and initialize counters for files and file types
Dim i, nEXE, nDLL, nOCX
i = 0
nEXE = 0: nDLL = 0: nOCX = 0

' Loop through components to get the number of files and individual file names
Dim sFileName
Dim pComp, pFile
```

```
For Each pComp In pProj.ISWiComponents
  For Each pFile In pComp.ISWiFiles
    i = i + 1
    sFileName = pFile.DisplayName
    If Instr (1, sFileName, ".exe", vbTextCompare) > 0 Then
      nEXE = nEXE + 1
    ElseIf Instr (1, sFileName, ".dll", vbTextCompare) > 0 Then
      nDLL = nDLL + 1
    ElseIf Instr (1, sFileName, ".ocx", vbTextCompare) > 0 Then
      nOCX = nOCX + 1
    End If
  Next
Next

sItems = sItems & "Number of files: " & i & vbNewLine
sItems = sItems & vbTab & "EXEs: " & nEXE & vbNewLine
sItems = sItems & vbTab & "DLLs: " & nDLL & vbNewLine
sItems = sItems & vbTab & "OCXs: " & nOCX & vbNewLine

' Display report
Wscript.Echo sItems

' Close project
pProj.CloseProject
```

Using Source Code Control at the Command Line

Access to installation projects stored in a source control system can be achieved through a combination of command-line calls to your source control application and limited use of the InstallShield Automation Interface—which gives you command-line-type access to much of InstallShield's functionality. Checking in, checking out, and getting the latest version of the project is handled through your source control's command-line support, if any.

Below are two examples of how you might use this functionality. Both examples use Microsoft Visual SourceSafe as the source control application and are written using VBScript.

You can copy and paste either example into a .vbs file, change the paths so that they are valid for your environment, and run them.



Note • *With earlier versions of InstallShield products, you were required to convert the source control (.isv) files to InstallShield project (.ism) files, and then back to .isv files. However, with the latest versions of InstallShield products, this is no longer necessary because you can save .ism files as text files for your source control application.*

Getting the Latest Version and Building Your Installation

```
Const VSSFLAG_USERRONO   = 1
Const VSSFLAG_TIMEMOD    = 8
Const VSSFLAG_REPREPLACE = 128

Const PROJECT_SCC_INI_LOC = "\\Server\srcsafe.ini"
Const PROJECT_SCC_FOLDER = "$/MyFiles/"
Const PROJECT_SCC_BASE_NAME = "MyProject"
Const PROJECT_SCC_LOCAL_FOLDER = "C:\Project"

' Create a ref to Microsoft Visual SourceSafe
```

```
    Set VSS = CreateObject("SourceSafe")
    ' Point to the VSS database
    VSS.open PROJECT_SCC_INI_LOC

    ' Get the project file
    Set VSSISVFile = VSS.VSSItem (PROJECT_SCC_FOLDER + PROJECT_SCC_BASE_NAME + ".ism")
    VSSISVFile.Get PROJECT_SCC_LOCAL_FOLDER + PROJECT_SCC_BASE_NAME + ".ism", SSFLAG_TIMEMOD +
VSSFLAG_USERRONO + VSSFLAG_REPREPLACE

    'Get all remaining files
    Set VSSIDTFolder = VSS.VSSItem (PROJECT_SCC_FOLDER + PROJECT_SCC_BASE_NAME)
    VSSIDTFolder.LocalSpec = PROJECT_SCC_LOCAL_FOLDER + PROJECT_SCC_BASE_NAME
    For Each VSSObj In VSSIDTFolder.Items(False)
        VSSObj.Get , VSSFLAG_TIMEMOD + VSSFLAG_USERRONO + VSSFLAG_REPREPLACE
    Next

    strFileBasePath = PROJECT_SCC_LOCAL_FOLDER + PROJECT_SCC_BASE_NAME + ".ism"
    strCmdLine = "ISCmdBld.exe -p """ + strFileBasePath + ".ism"""
    ' Build your installation
    Set wshshell = CreateObject("Wscript.Shell")
    RunCmdLine = wshshell.Run(strCmdLine, 1, True)
```

Checking Out, Making Changes, and Checking In

```
    Const VSSFLAG_USERRONO = 1
    Const VSSITEM_FILE     = 1

    Const PROJECT_SCC_INI_LOC = "\\Server\srcsafe.ini"
    Const PROJECT_SCC_FOLDER = "$/MyFiles/"
    Const PROJECT_SCC_BASE_NAME = "MyProject"
    Const PROJECT_SCC_LOCAL_FOLDER = "C:\Project"

    ' Create a ref to Microsoft Visual SourceSafe
    Set VSS = CreateObject("SourceSafe")
    ' Point to the VSS database
    VSS.open PROJECT_SCC_INI_LOC

    ' Check out the project file
    Set VSSISVFile = VSS.VSSItem (PROJECT_SCC_FOLDER + PROJECT_SCC_BASE_NAME + ".ism")
    VSSISVFile.CheckOut , PROJECT_SCC_LOCAL_FOLDER + PROJECT_SCC_BASE_NAME + ".ism", VSSFLAG_USERRONO

    'Check out all remaining files
    Set VSSIDTFolder = VSS.VSSItem (PROJECT_SCC_FOLDER + PROJECT_SCC_BASE_NAME)
    VSSIDTFolder.LocalSpec = PROJECT_SCC_LOCAL_FOLDER + PROJECT_SCC_BASE_NAME
    For Each VSSObj In VSSIDTFolder.Items(False)
        If VSSObj.Type = VSSITEM_FILE Then
            VSSObj.CheckOut , , VSSFLAG_USERRONO
        End If
    Next

    ' Create a reference to the InstallShield Automation Interface
    Set m_ISWiProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
    strFileBasePath = PROJECT_SCC_LOCAL_FOLDER + PROJECT_SCC_BASE_NAME + ".ism"
    ' Open your project
    m_ISWiProject.OpenProject strFileBasePath
    ' Add a feature
    m_ISWiProject.AddFeature "Robofeature1"
    ' Save the project
    m_ISWiProject.SaveProject
```

```
' Close the project
m_ISWiProject.CloseProject

' Check in the project file
Set VSSISVFile = VSS.VSSItem (PROJECT_SCC_FOLDER + PROJECT_SCC_BASE_NAME + ".ism")
VSSISVFile.CheckIn

'Check in all remaining files
Set VSSIDTFolder = VSS.VSSItem (PROJECT_SCC_FOLDER + PROJECT_SCC_BASE_NAME)
For Each VSSObj In VSSIDTFolder.Items(False)
    If VSSObj.Type = VSSITEM_FILE Then
        VSSObj.CheckIn "Check In Comment"
    End If
Next
```

Using Build Status Events

The automation interface includes build status events—ProgressIncrement, ProgressMax, and StatusMessage—that you can use to show the progress of the build and see status updates.

Example for Basic MSI, InstallScript, and InstallScript MSI Projects

The following Visual Basic 6 code demonstrates how to use these events in a build script in Basic MSI, InstallScript, and InstallScript MSI projects.

```
Public WithEvents pISWiRelease As ISWiAutoAutomation Interface Version.ISWiRelease

Private Sub Foo()
    Dim pISWiProject As IswiAutoAutomation Interface Version.ISWiProject
    Set pISWiProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
    pISWiProject.OpenProject "C:\InstallShield 2016 Projects\My Project Name-1.ism", False
    Set pISWiRelease = pISWiProject.ISWiProductConfigs("Product Configuration 1").ISWiReleases("Release
1")
    pISWiRelease.Build
    pISWiProject.CloseProject
    Set pISWiRelease = Nothing
    Set pISWiProject = Nothing
End Sub

Private Sub pISWiRelease_ProgressIncrement(ByVal lIncrement As Long, pbCancel As Boolean)
    ' Place your code here
End Sub

Private Sub pISWiRelease_ProgressMax(ByVal lMax As Long, pbCancel As Boolean)
    ' Place your code here
End Sub

Private Sub pISWiRelease_StatusMessage(ByVal sMessage As String, pbCancel As Boolean)
    ' Place your code here
End Sub
```

Example for Advanced UI and Suite/Advanced UI Projects

The following Visual Basic 6 code demonstrates how to use these events in a build script in Advanced UI and Suite/Advanced UI projects.

```
Public WithEvents pISWiSuiteRelease As ISWiAutoSuiteAutomation Interface Version.ISWiSuiteRelease

Private Sub Foo()
    Dim pISWiProject As ISWiAutoSuiteAutomation Interface Version.ISWiProject
    Set pISWiProject = CreateObject("ISWiAutoSuiteAutomation Interface Version.ISWiProject")
    pISWiProject.OpenProject "C:\InstallShield 2016 Projects\My Project Name-1.issuite", False
    Set pISWiRelease = pISWiProject.ISWiSuiteReleases("Release 1")
    pISWiSuiteRelease.Build
    pISWiProject.CloseProject
    Set pISWiSuiteRelease = Nothing
    Set pISWiProject = Nothing
End Sub

Private Sub pISWiSuiteRelease_ProgressIncrement(ByVal lIncrement As Long, pbCancel As Boolean)
    ' Place your code here
End Sub

Private Sub pISWiSuiteRelease_ProgressMax(ByVal lMax As Long, pbCancel As Boolean)
    ' Place your code here
End Sub

Private Sub pISWiSuiteRelease_StatusMessage(ByVal sMessage As String, pbCancel As Boolean)
    ' Place your code here
End Sub
```

Using the Automation Interface on a 64-Bit System

The automation interface is a 32-bit interface; therefore, it must be loaded from a 32-bit process. If you are using the automation interface on a 64-bit machine, you may need to load the automation interface through a 32-bit executable file.

For example, if you are using VBScript with the automation interface, you may need to launch **cscript.exe** from the 32-bit system folder (SysWow64). Otherwise, the 64-bit scripting host may encounter an error such as the following one when creating the automation object:

Microsoft VBScript runtime error: ActiveX component can't create object: 'ISWiAutoAutomation Interface Version.ISWiProject'

12

Reference

Reference information for InstallShield is organized into the following sections:

Table 12-1 • Reference Selections

Section	Description
Menu, Toolbar, and Window Reference	Describes the various components of the InstallShield user interface, including menus, toolbars, and windows.
Dialog Box Reference	Contains reference information on each of the dialog boxes that are displayed in InstallShield.
Wizard Reference	Provides details about each of the wizards that are available in InstallShield.
View Reference	Describes each of the views that are displayed in InstallShield.
InstallShield Prerequisite Editor Reference	Introduces you to the InstallShield Prerequisite Editor, which is available with InstallShield. Use this tool to create InstallShield prerequisites that you can include in your projects, or to edit existing ones.
Errors and Warnings	Provides information about error codes and warnings that might occur when you create, compile, build, or run your installation. This section also includes reference information about errors and warnings that may occur when you migrate a project from an earlier version of an InstallShield product to the latest version.
Automation Interface	Contains reference material that describes the objects, methods, properties, and collections that you can use to create, modify, and build a project through the automation interface.

Table 12-1 • Reference Selections (cont.)

Section	Description
Object Reference for Expressions in Advanced UI and Suite/Advanced UI Projects	Explains each of the objects that you can use to embed object expressions in various settings of an Advanced UI or Suite/Advanced UI project. Object expressions enable you to search target systems for information about a file, registry entry, or other item. At run time, the installation expands the value of the expression and searches the target system.
InstallShield Custom Action Reference	Explains each of the custom actions that are available in InstallShield.
Command-Line Tools	Introduces tools that you can use from the command line to perform tasks such as building a release and running an installation.
InstallScript Language Reference	Provides in-depth information about functions, data types, event handlers, operators, and other aspects of InstallScript, a simple but powerful programming language that you can use to design your installations. This section also contains sample code that you can use in your script file.

Menu, Toolbar, and Window Reference

This section describes the various components of the InstallShield user interface, including menus, toolbars, and windows.

Menus

The menus in InstallShield are located on the menu bar, which is at the top of the InstallShield user interface. Each menu contains a list of commands. Some of these commands have icons next to them so that you can quickly associate the command with the icon.

Each of the menus in InstallShield is described in this section:

- File
- Edit
- View
- Go
- Project
- Build
- Tools
- Window
- Help

File Menu

The following table lists the File menu commands, as well as associated keyboard shortcuts and icons.

Table 12-1 • File Menu Commands

Command	Shortcut	Icon	Description
New	Ctrl+N		Creates a new installation project.
Open	Ctrl+O		Opens an existing installation project.
Close			Closes the current project.
Save	Ctrl+S		Saves the current project.
Save As			Enables you to save the current project file with a new name and location.
Print	Ctrl+P		Prints the script file that is displayed in the active script window.

Table 12-1 • File Menu Commands (cont.)

Command	Shortcut	Icon	Description
1, 2, 3, 4			Opens one of the most recently accessed projects.
Exit			Closes the current project and closes InstallShield.

Edit Menu

The following table lists the Edit menu commands, as well as associated keyboard shortcuts and icons.

Table 12-2 • Edit Menu Commands

Command	Shortcut	Icon	Description
Undo	Ctrl+Z		Undoes the last action performed.
Redo	Ctrl+Y		Reverses the last action that was performed with the Undo command.
Cut	Ctrl+X		Removes the currently selected text and places it on the Clipboard.
Copy	Ctrl+C		Copies the currently selected text to the Clipboard.
Paste	Ctrl+V		Inserts the contents of the Clipboard at the insertion point, and replaces any selected text.
Find	Ctrl+F		Searches for the specified text, file, or folder.
Replace	Ctrl+H		Searches for and replaces the specified text.
Find Next	F3		Searches for the next occurrence of the specified text.
Go To	Ctrl+G		Moves the insertion point to the specified line number in the active script window.
Find String ID in Project			Searches the entire installation project for occurrences of a specified string entry.
Repeat			Opens the Set Repeat Count dialog box, where you can specify the total number of times that the next character you type should be inserted.

Table 12-2 • Edit Menu Commands (cont.)

Command	Shortcut	Icon	Description
Insert			Provides two commands: • **InstallScript Function**— Opens the Function Wizard, which automates the process of adding a function call to the active script displayed in the InstallScript view. • **String Entry**— Opens the Select String Dialog Box, which lets you select a string entry that you want to add to the active script displayed in the InstallScript view.

View Menu

The following table lists the View menu commands, as well as associated keyboard shortcuts and icons.

Table 12-3 • View Menu Commands

Command	Shortcut	Icon	Description
Output Window			Shows or hides the Output window.
View List	F4		Shows or hides the View List.
View Bar			Shows or hides the View Bar.
Header Bar			Shows or hides the header bar, which has the tabs for the Start Page, the Project Assistant, and the Installation Designer.
Standard			Shows or hides the Standard toolbar.
MSI Debugger			Shows or hides the MSI Debugger toolbar.
Layout			Shows or hides the Layout toolbar.
Controls			Shows or hides the Controls toolbar.
Status Bar			Shows or hides the status bar.
Watch			Shows or hides the Watch window in the InstallScript Debugger.
Variable			Shows or hides the Variable window in the InstallScript Debugger.
Maximize	Ctrl+M		Changes the width of the script or dialog editor window that is currently open.
Project Assistant			Displays the Project Assistant.

Table 12-3 • View Menu Commands (cont.)

Command	Shortcut	Icon	Description
Microsoft App-V			Displays the Microsoft App-V tab.
VMware ThinApp			Displays the VMware ThinApp tab.

Go Menu

The following table lists the Go menu commands, as well as associated keyboard shortcuts and icons. Some of the view-related commands are not available from the Go menu, depending on which project type you have open in InstallShield.

Table 12-4 • Go Menu Commands

Command	Shortcut	Icon	Description
Previous View	ALT+UP ARROW		Takes you to the view directly above the current view as shown in the View List.
Next View	ALT+DOWN ARROW		Takes you to the view directly below the current view as shown in the View List.
Back	ALT+LEFT ARROW		Takes you to the view that you last visited in the history of your view selections. You can use this multiple times, as long as there are multiple entries in your view history.
Forward	ALT+RIGHT ARROW		Takes you to the next view in the history of your view selections. You can continue until you reach the view you were at when you first clicked Back.
Start Page			Takes you to the Start Page.
Help			Takes you to the Help view.
Project Assistant			Displays the Project Assistant.
Microsoft App-V			Displays the Microsoft App-V tab.
VMware ThinApp			Displays the VMware ThinApp tab.
Installation Information			Enables you to open the General Information view. In Basic MSI and InstallScript MSI projects, this menu also enables you to open the Update Notifications view.
Organization			Enables you to display the Setup Design, Features, Components, and Setup Types.

Table 12-4 • Go Menu Commands (cont.)

Command	Shortcut	Icon	Description
Application Data			Enables you to display the Files and Folders, Redistributables (or Objects in InstallScript projects), and Prerequisites views.
System Configuration			Enables you to display the Shortcuts, Registry, ODBC Resources, INI File Changes, Environment Variables, XML File Changes, and Text File Changes views.
Server Configuration			Enables you to display the Internet Information Services, Component Services, and SQL Scripts views.
Behavior and Logic			Enables you to display the InstallScript, Custom Actions and Sequences (or Custom Actions), Support Files (or Support Files/Billboards in InstallScript and InstallScript MSI projects), System Search, and Property Manager views.
User Interface			Enables you to display the Dialogs, Billboards, and String Editor views.
Media			Enables you to display the Path Variables, Upgrades, Releases, and Patch Design views.
Additional Tools			Enables you to display the Dependency Scanners, MSI Debugger, and Direct Editor views.
Patch Settings			Enables you to display the General Information, Files, Registry, and Patch Variables views.

Project Menu

The following table lists the Project menu commands, as well as associated icons.

Table 12-5 • Project Menu Commands

Command	Icon	Description
Scan Project using FlexNet Code Aware		Launches FlexNet Code Aware to run a scan of the open InstallShield project to help expose and assess IP and security vulnerability risks associated with your project. FlexNet Code Aware is an automated open source risk assessment and package discovery solution that requires a separate license from InstallShield, There is also trial/evaluation version. For more information, refer to the FlexNet Code Aware product page of the Flexera Software website.  ***Note*** • *If you do not have an InstallShield project open, this menu item is disabled.*
Visual Studio Deployment Project Import Wizard		Launches the wizard that lets you import a Visual Studio setup or merge module project into an InstallShield project.
Export Components Wizard		Opens the Export Components Wizard, which simplifies the process of exporting a component from a file to a new or existing project.
Device Driver Wizard		Opens the Device Driver Wizard, which guides you through the process of adding a device driver to your installation.
Reg-Free COM Wizard		Opens the Reg-Free COM Wizard, which guides you through the process of adding a COM manifest file to your project or configuring an existing manifest file.
Perform Static Scan		Opens the Static Scanning Wizard, which scans the files already added to your project for any dependencies that they require.
Perform Dynamic Scan		Opens the Dynamic Scanning Wizard, which monitors a running executable for all .dll and .ocx dependencies files and automatically adds them to your project.
Convert Source Paths		Opens the Convert Source Paths Wizard, which lets you convert existing hard-coded paths with path variables, thereby enhancing the portability of your installation project.

Table 12-5 • Project Menu Commands (cont.)

Command	Icon	Description
Project Converters		Provides several commands: • **Convert to InstallScript MSI Project**—Converts your Basic MSI project to an InstallScript MSI project. For more information, see Converting a Basic MSI Project to an InstallScript MSI Project. • **Convert to InstallScript Project**—Opens the Convert InstallScript MSI Dialog Box to help you convert your InstallScript MSI project to an InstallScript project.
Source Control		Provides several commands: • **Get Latest Version**— Copies the most recent version of the current project into your project folder. The project placed in your project folder is read-only. • **Check Out**— Places a writable copy of the project into your project folder, enabling you to modify it. • **Check In**— Stores your changes to the current project in your source control application. • **Undo Check Out**— Undoes your checkout, canceling your changes both in the source control application and in your project folder. Your project reverts to the way it was before you checked it out. • **Add to Source Control**— Adds the current project to your source control application. • **Unlink from Source Control**— Breaks the link between the current project and your source control application. • **Show History**— Displays a record of changes to your project since it was initially added to your source control application. While viewing the history, you can return to any point in your project's history and recover the file as it existed at that point. • **Show Differences**— Compares two files, then shows you the differences, if any. • **Refresh Status**— Refreshes the source control status. • **Launch Source Control Program**— Opens your source code control application.
Settings		Opens the Project Settings Dialog Box.

Build Menu

The following table lists the Build menu commands, as well as associated keyboard shortcuts and icons.

Table 12-6 • Build Menu Commands

Command	Shortcut	Icon	Description
Release Wizard			Opens the Release Wizard, which lets you specify the settings for a release and build it.
Compile	Ctrl+F7		Compiles your script file.
Build	F7		Builds your release with default settings, or—if you have already built your release—rebuilds your release with your most recently saved settings.
Build Tables Only	SHIFT+F7		Rebuilds only the .msi file tables of your installation to enable you to see your changes take effect more quickly than with a complete rebuild.
Build Tables & Refresh Files	ALT+F7		 ***Project*** • *This command is available for Windows Installer–based projects.* Rebuilds only the .msi file tables and updates only the modified files of your installation. This type of build works only when the media is an uncompressed network image.
Refresh Build	ALT+F7		 ***Project*** • *This command is available for InstallScript projects.* Rebuilds the active media. This command regenerates only those items that you changed since the media was last built.
Batch Build			Lets you rebuild multiple releases with different configurations simultaneously.
Stop Build	Ctrl+Break		Cancels the current build process.
Test	Ctrl+T		Enables you to run through the user interface portion of your installation without making any changes to your system. All custom actions are executed.
Debug	F5		Starts debugging.
Run	Ctrl+F5		Enables you to run your completed installation without leaving InstallShield.

Table 12-6 • Build Menu Commands (cont.)

Command	Shortcut	Icon	Description
Uninstall			Uninstalls the most recently run release.  ***Project*** • *This command is available for the following project types:* • *Basic MSI* • *InstallScript MSI*
Run Package			Enables you to run the installation package. This command is available only for InstallScript projects with the Network Image media type and with all files compressed into an .exe file.
Debug Package			Enables you to debug the installation package. This command is available only for InstallScript projects with the Network Image media type and with all files compressed into an .exe file.
Run from Web			Displays the One-Click Install Web page, which lets you run your completed installation without leaving InstallShield.
MSI Debugger			Provides several commands: • **Start MSI Debugging**—Starts debugging in the MSI Debugger view. • **Toggle Breakpoint**—F9 or —Sets or removes the breakpoint in the line that has the insertion point. • **Remove All Breakpoints**—SHIFT+F9 or —Removes all breakpoints set in the debugger. • **Step Over**—F10 or —Steps through the current action or dialog and continues the execution in the debugger. • **Step Into**—F11 or —Steps into the current custom action in the debugger. • **Stop MSI Debugging**—SHIFT+F5 or —Stops debugging in the MSI Debugger view.

Table 12-6 • Build Menu Commands (cont.)

Command	Shortcut	Icon	Description
Validate			Provides several commands: • **Upgrade Validation Wizard**—Opens the Upgrade Validation Wizard, which performs a set of tests to determine if your Basic MSI or InstallScript MSI installation will properly upgrade earlier versions of your product. • **Browse for a New Validation Module (.cub)**—Lets you browse to select a new or custom validation (.cub) file. • **InstallShield Validation Suite for Windows 8**—This suite consists of a number of InstallShield internal consistency evaluators (ISICEs) that help you identify issues that may make your installation behave unexpectedly on Windows 8 systems. This suite checks for issues that may not be revealed in the Full MSI Validation Suite. • **InstallShield Merge Module Validation Suite for Windows 8** or **InstallShield Merge Module Validation Suite for Windows 7**—These suites consist of a number of InstallShield internal consistency evaluators (ISICEs) that help you identify issues that may make your merge module behave unexpectedly on Windows 8 (or Windows 7) systems. This suite checks for issues that may not be revealed in the Merge Module Validation Suite. • **InstallShield Certified for Windows Vista Validation Suite**—This suite consists of a number of InstallShield internal consistency evaluators (ISICEs) that help you identify issues that may make your installation behave unexpectedly on Windows Vista systems. This suite checks for issues that may not be revealed in the Full MSI Validation Suite. • **InstallShield Certified for Windows Vista Merge Module Validation Suite**—This suite consists of a number of InstallShield internal consistency evaluators (ISICEs) that help you identify issues that may make your merge module behave unexpectedly on Windows Vista systems. This suite checks for issues that may not be revealed in the Merge Module Validation Suite.

Table 12-6 • Build Menu Commands (cont.)

Command	Shortcut	Icon	Description
Validate (cont.)			• **InstallShield Best Practice Suite**— Scans your Basic MSI or InstallScript MSI installation to determine whether it meets best-practice guidelines. ***Edition*** • *The InstallShield Best Practice Suite is available in the Premier edition of InstallShield.* • **Full MSI Validation Suite**— Scans your Basic MSI or InstallScript MSI installation to determine whether it is valid. Your built .msi package is compared against the ICE rules. • **Windows 2000 Logo Validation Suite**— Scans your Basic MSI or InstallScript MSI installation to determine whether it meets the Designed for Windows 2000 logo requirements. • **Windows XP Logo Validation Suite**— Scans your Basic MSI or InstallScript MSI installation to determine whether it meets the Designed for Windows XP logo requirements. • **Windows Vista Logo Validation Suite**— Scans your Basic MSI or InstallScript MSI installation to determine whether it meets the Certified for Windows Vista logo requirements. • **Merge Module Validation Suite**— Scans your merge module package. The Merge Module Validation tool is the merge module version of the Full MSI Validation Suite. Your built merge module is compared against ICE rules to help you determine whether it will function properly. • **InstallShield Virtualization Suitability Suite**—Scans an .msi package running all tests related to virtualization suitability. Any tests that found issues are listed in a report, and for each issue, an associated column in the report indicates applicability to known virtualization formats (App-V v5, Server App-V, App-V-v4, ThinApp, or XenApp). • **InstallShield UWP App Suitability Suite**—Scans an .msi package for signs of items that are unsuitable for the UWP app package (.appx) format. Any tests that found issues are listed in a report and for each issue, an associated column in the report indicates applicability to the known UWP app variants (UWA, Desktop, Store, WSA, or Nano). • **Recent Validation Modules**—Displays a list of recently used validation (.cub) files. To learn more, see Validating Projects and Viewing Validation Results.
Settings	ALT+F6		Opens the Settings dialog box, which enables you to set preferences for how InstallShield builds your installation.

Tools Menu

The following table lists the Tools menu commands, as well as associated icons.

Table 12-7 • Tools Menu Commands

Command	Icon	Description
Open Release Folder		Launches Windows Explorer, open in the DISK1 folder of the current release. If there is no release or if a release has not been built, Windows Explorer opens to your default project location.
Add New Language		Opens the New Language Wizard, which enables you to add support for almost any language to one or more projects.
Create/Apply Transform		Opens the Transform Wizard, which walks you through the steps of creating and applying transforms for your installation projects.
MSI Log Analyzer		Launches the InstallShield MSI Log Analyzer, a utility that enables you to generate reports based on an analysis of Windows Installer log files.
MSI Command-Line Builder		Launches the InstallShield MSI Command-Line Builder. This tool enables you to quickly build complex command-line strings for executing Windows Installer–related tasks.
Difference		Provides several commands: • **Compare to File**—Opens a dialog box that lets you specify which project should be compared with the currently open project. • **End Compare**—Ends the previously started compare session. • **InstallShield MSI Diff**—Opens InstallShield MSI Diff, a tool that enables you to compare Windows Installer tables in two different Windows Installer databases. It also enables you to apply transforms and patches to Windows Installer packages and then see how the database tables are affected. To learn more, see InstallShield MSI Diff.

Table 12-7 • Tools Menu Commands (cont.)

Command	Icon	Description
InstallScript		Provides several commands: • **Standard Dialog Sampler**— Opens the Standard Dialog Sampler, which enables you to view examples of end-user dialogs. The sample dialogs are displayed without a skin. • **Skinned Dialog Sampler**— Opens the Skinned Dialog Sampler, which enables you to view examples of end-user dialogs. The sample dialogs are displayed with the skin that is selected in the Dialogs view. • **Cabinet File Viewer**— Opens the Cabinet File Viewer, a tool that lets you select a cabinet (.cab) file and view its compressed files, file groups, components, and setup types, as well as the properties of those items. It also lets you extract files from the cabinet file. • **Log File Viewer**— Opens the Log File Viewer, a tool that enables you to view the installation log file. The log file is created when your installation is first run and it is maintained when your installation is running in maintenance mode. You can use the log file to determine the current state of each component in the installation.
Check for Updates		Checks for service packs and other updates to InstallShield. If updates are available, you can select which ones you would like to download and install.
Redistributable Downloader		Launches the Redistributable Downloader Wizard to quickly download third-party redistributables, merge modules, and other files to your local machine.
Prerequisite Editor		Opens the InstallShield Prerequisite Editor, which enables you to create or modify InstallShield prerequisites that you can include in your installation project.
Customize		Displays the Customize dialog box, which enables you to customize the toolbars displayed in the InstallShield user interface.
Options		Displays the Options dialog box, which enables you to specify preferences for creating projects and working in InstallShield.

Window Menu

The following table lists the Window menu commands, as well as associated keyboard shortcuts and icons.

Table 12-8 • Window Menu Commands

Command	Shortcut	Icon	Description
Next	Ctrl+Tab		Opens the InstallScript view and displays the next script file in your project as shown at the bottom of the Window menu.
Previous	Ctrl+Shift+Tab		Opens the InstallScript view and displays the next script file in your project as shown at the bottom of the Window menu.
1, 2, 3			Opens the InstallScript view and displays one of the script files included your project.

Help Menu

The following table lists the Help menu commands, as well as associated icons.

Table 12-9 • Help Menu Commands

Command	Icon	Description
Contents		Displays the Contents tab of the online help library.
Index		Displays the Index tab of the online help library.
Search		Displays the Search tab of the online help library.
Support Central		Displays Support Central on the Web.
InstallShield Community		Displays the Community on the Web.
Release Notes		Displays InstallShield release notes.
Flexera Software on the Web		Connects to the Flexera Software Web site.
Help View		Displays the Help view.
About InstallShield		Displays the About InstallShield dialog box, where you can find version information and register InstallShield.

Toolbars

The InstallShield user interface offers several toolbars that give you quick access to frequently used menu items and provide graphical tools for use in the Dialog Editor.

Built-in Toolbars

The following toolbars are available in the InstallShield interface. When you open InstallShield, only the Standard toolbar is displayed near the top of the user interface. The Control and Dialog Layout toolbars open when you work in the Dialogs view and disappear when you leave that view.

All toolbars can be resized and repositioned, and docked and undocked.

- Standard Toolbar
- Controls Toolbar
- Layout Toolbar
- MSI Debugger Toolbar

Standard Toolbar

Below is a description of all the buttons on the Standard toolbar.

Table 12-10 • Standard Toolbar Buttons

Button	Name	Description
	New Project	Launches the New Project dialog box, which enables you to select a project type and begin a new project.
	Open	Enables you to open project files or convert installation package (.msi) files to an InstallShield installation project (.ism) file.
	Save	Saves changes to the project file.
	Undo	Click the Undo button in the Dialog Editor or the Script Editor to reverse the last change you made.
	Redo	Click the Redo button in the Dialog Editor or the Script Editor to restore the last action you reversed by clicking the Undo button.
	Viewbar	Hides or shows the viewbar, which appears on the left side of the interface.
	View List	Hides or shows the View List, which shows all the views available in the InstallShield interface.  ***Note*** • *You can also press F4 to hide or show the View List.*
	Previous View	Displays the view directly above the current view, as shown in the View List.

Table 12-10 • Standard Toolbar Buttons (cont.)

Button	Name	Description
	Next View	Displays the view directly below the current view, as shown in the View List.
	Back	Displays the view that you last visited in the history of your view selections. You can click this button multiple times, if there are multiple entries in your view history.
	Forward	Displays the next view in the history of your view selections. You can continue clicking this button until you reach the view in which you first clicked the Back button.
	Insert InstallScript Function	Launches the Function Wizard, which eases the process of writing built-in InstallScript functions.
	Release Wizard	Launches the Release Wizard, which turns a project into either a shippable Windows Installer setup package or completed merge module. The options you select in the wizard determine the layout and contents of your release.
	Compile	Compiles the InstallScript files in your project without building the entire setup.
	Build	Builds your release with default settings, or—if you have already built your release—rebuilds your release with your most recently saved settings.
	Stop Build	Cancels the current build process.
	Test User Interface	Runs through the user interface portion of your installation project without making any changes to your system. All custom actions are executed.  ***Project*** • *The Test button is available for the following project types:* • Basic MSI • InstallScript MSI
	Run	Runs your completed installation project.  ***Project*** • *For Basic MSI and InstallScript MSI projects—If you selected the* ***Uninstall the product automatically before installing or debugging*** *check box, InstallShield uninstalls your product before rerunning the installation or before starting the debugger. This check box is available on the Preferences tab of the Options dialog box.*

Table 12-10 • Standard Toolbar Buttons (cont.)

Button	Name	Description
	Uninstall	Uninstalls the most recently run release.  ***Project*** • *This button is available for the following project types:* • *Basic MSI* • *InstallScript MSI*
	Debug	Runs the appropriate debugger, depending on the type of installation you are creating. If your project is InstallScript or InstallScript MSI, clicking Debug launches the InstallScript Debugger. If your project is Basic MSI or a Transform, clicking Debug launches the MSI Debugger.  ***Project*** • *For Basic MSI and InstallScript MSI projects—If you selected the* ***Uninstall the product automatically before installing or debugging*** *check box, InstallShield uninstalls your product before rerunning the installation or before starting the debugger. This check box is available on the Preferences tab of the Options dialog box.*
	Open Release Folder	Launches Windows Explorer, open in the DISK1 folder of the current release. If there is no release or if a release has not been built, Windows Explorer opens to your default project location.
	Scan Project using FlexNet Code Aware	Launches FlexNet Code Aware to run a scan of the open InstallShield project to help expose and assess IP and security vulnerability risks associated with your project. FlexNet Code Aware is an automated open source risk assessment and package discovery solution that requires a separate license from InstallShield, There is also trial/evaluation version. For more information, refer to the FlexNet Code Aware product page of the Flexera Software website.
	Help View	Displays the IDE Help view where you can find answers to many of your questions regarding InstallShield.

Controls Toolbar

The Controls toolbar provides every standard control that is available in a run-time dialog. This toolbar is displayed when you edit a dialog's layout in the Dialog Editor.

Task ***To draw a control on a dialog:***

Click the control's button, click anywhere on the face of the dialog, drag the mouse pointer while continuing to press the mouse button, and then release the button when you have reached the intended dimensions of the control.

Table 12-11 • Controls Toolbar Buttons

Button	Name	Description
	Select Tool	Lets you give focus to a control on a dialog. You can then move, resize, delete, or align the control.
	Check Box	Lets you draw a check box control on a dialog.
	Push Button	Lets you draw a push button control on a dialog.
	Edit Field	Lets you draw an edit field control on a dialog.
	Combo Box	Lets you draw a combo box control on a dialog.
	Text Area	Lets you draw a text area control on a dialog.
	List Box	Lets you draw a list box control on a dialog.
	Radio Button	Lets you draw a radio button control on a dialog. A radio button must be added to a radio button group control.
	Bitmap	Lets you draw a bitmap control on a dialog.
	Group Box	Lets you draw a group box control on a dialog.
	Billboard	Lets you draw a billboard control on a dialog.
	Line	Lets you draw a line control on a dialog.
	Radio Button Group	Lets you draw a radio button group control on a dialog.
	Selection Tree	Lets you draw a selection tree control on a dialog.
	Progress Bar	Lets you draw a progress bar control on a dialog.
	List View	Lets you draw a list view control on a dialog.

Table 12-11 • Controls Toolbar Buttons (cont.)

Button	Name	Description
	Scrollable Text	Lets you draw a scrollable text control on a dialog.
	Icon	Lets you draw an icon control on a dialog.
	Directory List	Lets you draw a directory list control on a dialog.
	Directory Combo	Lets you draw a directory combo control on a dialog.
	Volume Cost List	Lets you draw a volume cost list control on a dialog.
	Volume Select Combo	Lets you draw a volume select combo control on a dialog.
	Masked Edit	Lets you draw a masked edit control on a dialog.
	Path Edit	Lets you draw a path edit control on a dialog.
	Hyperlink	Lets you draw a hyperlink control. This control displays an HTML link to an address; when the end user clicks the link, the page opens in the default browser.

Layout Toolbar

The Layout toolbar provides several tools for aligning and resizing controls in a dialog. This toolbar is displayed when you edit a dialog's layout in the Dialog Editor.

Most buttons on this toolbar are enabled only when you have selected more than one control. To select multiple controls, select one, press the SHIFT key, and select the other; or hold the mouse button down and drag the cursor across several controls. The Layout toolbar contains the following buttons:

Table 12-12 • Layout Toolbar Buttons

Button	Name	Description
	Align Left	Select two or more controls and click the Align Left button to align all of the selected controls with the left edge of the leftmost control.
	Align Right	Select two or more controls and click the Align Right button to align all of the selected controls with the right edge of the rightmost control.
	Align Top	Select two or more controls and click the Align Top button to align all of the selected controls with the top edge of the uppermost control.

Table 12-12 • Layout Toolbar Buttons (cont.)

Button	Name	Description
	Align Bottom	Select two or more controls and click the Align Bottom button to align all of the selected controls with the bottom edge of the lowermost control.
	Center Vertical	Select a control and click the Center Vertical button to reposition the element in the vertical (x-axis) center of the dialog.
	Center Horizontal	Select a control and click the Center Horizontal button to reposition the element in the horizontal (y-axis) center of the dialog.
	Space Across	Select three or more controls on the dialog and click the Space Across button to have them evenly distributed horizontally across the dialog.
	Space Down	Select three or more controls on the dialog and click the Space Down button to have them evenly distributed vertically along the dialog.
	Make Same Width	Select two or more controls and click the Make Same Width button to resize the width of the smaller controls to the width of the largest control.
	Make Same Height	Select two or more controls and click the Make Same Height button to resize the height of the smaller controls to the height of the largest control.
	Make Same Size	Select two or more controls and click the Make Same Size button to make the smaller controls the exact size of the largest control.
	Bring to Front	Select a control and click the Bring to Front button to place it on top of any overlapping controls. See note below.
	Send to Back	Select a control and click the Send to Back button to place it behind any overlapping controls. See note below.
	Show Grid	Click the Show Grid button to place design-time grid lines on the dialog, or to remove the grid if it is present.



Note • *The Bring to Front and Send to Back actions are not saved when you close and reopen your project. These actions are for use during design only. For example, you might have a dialog that displays different controls based on a condition. You can use the Bring to Front and Send to Back actions to work on these different controls while you are designing your dialog.*

MSI Debugger Toolbar

Below is a description of all the buttons in the MSI Debugger toolbar.

Table 12-13 • MSI Debugger Toolbar Buttons

Button	Name	Description
	Toggle Breakpoint	Click this button to set a breakpoint on an action or dialog.
	MSI Debugging	Click this button to start debugging or, while debugging, to continue stepping through the setup.
	Step Over	Click this button to step over the current action or dialog.
	Step Into	Click this button to step into the current action and launch your registered debugger.
	Remove All Breakpoints	Click this button to clear all breakpoints you have set in the MSI Debugger.
	Stop MSI Debugger	Click this button to stop debugging.

Output Window

When you build a release for your project, the Output window opens if it is not already open, and it displays build information. It also provides other task-specific information. For example, if you validate your project, InstallShield displays validation results in the Output window. If you open a project that was created with an earlier version of InstallShield, the Output window shows upgrade information.

The following tabs appear in the Output window:

Table 12-14 • Output Window Tabs

Tab	Description
Build	Stores distribution output information and displays build output; a link to the output file saved as a text file is included.
Validate	Displays validation results.
Results	Displays project conversion information or details about testing changes to XML files.
Compile	Displays InstallScript information when you click Compile on the Build menu.
Tasks	Provides descriptions of error and warning codes when you build, compile, or validate your project; each error or warning code is a link to a Knowledge Base article. ***Tip*** • *You can right-click a code for an error or warning and select Direct Editor. The affected area associated with the error or warning will be highlighted in the Direct Editor.*
Distribute	Displays information about distributing a release to a virtual machine. To learn more, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Dialog Box Reference

Each of the dialog boxes available in the user interface of InstallShield is described in this section:

- .NET 1.1/2.0 Core Language Dialog Box
- .NET 1.1/2.0 Language Packs Dialog Box
- .NET Installer Class Arguments Dialog Box
- Add an Argument/Value Pair Dialog Box
- Add Existing Media Dialog Box
- Add Files for This Package Dialog Box
- Add MIME Type Dialog Box
- Add New Method Dialog Box
- Add New Property Dialog Box
- Add Predefined Search Dialog Box
- Add to Source Control Dialog Box
- Application Extension Mapping Dialog Box
- Application Mappings Dialog Box
- Associate Components Dialog Box
- Associate DIMs Dialog Box
- Batch Build Dialog Box
- Browse for Directory Dialog Box
- Browse for Directory/Shortcut Target Dialog Box
- Browse for File Dialog Box
- Browse for File to Run Dialog Box
- Browse for Shortcut Icon Dialog Box
- Browse for Tile Target Dialog Box
- Certificate Selection Dialog Box
- Check In Dialog Box
- Check Out Dialog Box
- Component Properties Dialog Box
- Condition Builder Dialog Box
- Content Source Path Dialog Box
- Convert InstallScript MSI Dialog Box
- Create a New Component Dialog Box

- Create a New Feature Dialog Box
- Custom Error Handling Dialog Box
- Custom Errors Dialog Box
- Customize Validation Settings Dialog Box
- Data Languages Dialog Box
- Delete Property Dialog Box
- Dependencies Dialog Box
- Dialog Images Dialog Box
- Digitally Sign Setup Dialog Box
- Differences Dialog Box
- Dynamic File Link Settings Dialog Box
- Edit Data Dialog Box
- Edit List Options Dialog Box
- Edit Registry Data Dialog Box
- Edit Virtual Machine Configurations Dialog Box
- Error Mapping Properties Dialog Box
- Exclusion Languages Dialog Box
- Export Tables Dialog Box
- Feature Condition Builder Dialog Box
- File Details Dialog Box
- File Properties Dialog Box
- File Properties Dialog Box (InstallScript Installation Projects)
- File Removal Properties Dialog Box
- Find and Replace Dialog Box
- Find String ID in Project Dialog Box
- Folder Properties Dialog Box
- Function Signature Dialog Box
- General Options - Advanced Dialog Box
- General Options - Other Disk Files Dialog Box
- History Dialog Box
- Import Dialog Dialog Box
- Import InstallScript Files Dialog Box
- Import SQL Script Files Dialog Box

- Insert Action Dialog Box
- InstallShield Prerequisite Properties Dialog Box
- Languages Dialog Box
- Link Type Dialog Box
- Logging Options for Windows Installer 4.0 and Later Dialog Box
- Lowercase Component Directory Warning
- Media File Properties Dialog Box
- Merge Module Configurable Values Dialog Box
- Merge Module Properties Dialog Box
- Method Browser Dialog Box
- Method Signature Dialog Box
- MIME Types Dialog Box
- Modify Dynamic Links Dialog Box
- Modify Property Dialog Box/Release Wizard—Platforms Panel
- Module Dependencies Dialog Box
- Module Exclusions Dialog Box
- MSI Log File Tab
- MSI Value Dialog Box
- MST SIS Settings Dialog Box
- Multi-Line String Value Dialog Box
- New Dependency Dialog Box
- New File Dialog Box
- New Project Dialog Box
- Operating Systems Dialog Box
- Options Dialog Box
- Other IIS Properties Dialog Box
- Other Window Styles Dialog Box
- Outputs Dialog Box
- Overwrite Dialog Box
- Patch Sequence Dialog Box
- Path Variable Overrides Dialog Box
- Path Variable Recommendation Dialog Box
- Permissions Dialog Boxes for Files and Directories

- Permissions Dialog Boxes for Registry Keys
- Platform Suites Dialog Box
- Platforms Dialog Box
- Postbuild Event Dialog Box
- Prebuild Event Dialog Box
- Precompression Event Dialog Box
- Prerequisite Condition Dialog Box
- Privileges Dialog Box
- Product Condition Builder Dialog Box
- Project Settings Dialog Box
- Rename Key Dialog Box
- Required Features Dialog Box
- Resolve Conflict Dialog Box
- Save As Dialog Box
- Script Conversion Dialog Box
- Script Editor Properties Dialog Box
- SCM Pre-shutdown Delay Dialog Box
- Security Descriptor Dialog Box
- Select File Dialog Box
- Select Media Location Dialog Box
- Select String Dialog Box
- Serial Number Violation Dialog Box
- Server Locations Dialog Box
- Service Options Dialog Box for a Time Delay of an Auto-Start Service
- Service Options Dialog Box for Running Recovery Actions
- Service SID Dialog Box
- Set File(s) URL Dialog Box
- Settings Dialog Box
- Setup Languages Dialog Box
- SQL Server Requirement Dialog Box
- Supersedence Dialog Box
- UI Languages Dialog Box
- Update Merge Module Search Path Dialog Box

.NET 1.1/2.0 Core Language Dialog Box



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Use the .NET 1.1/2.0 Core Language dialog box to select the .NET core language that you want to distribute. This is the language that is used while the .NET 1.1 core redistributable is installed.

If the 2.0 version of the .NET Framework is selected in the .NET Framework Version setting, the language options are all selected and disabled since they are all included with this version of the redistributable.

If you do not have a particular language installed on your system, the check box for that language is disabled in this dialog box.



***Tip** • To download one or more language versions of the .NET Framework redistributables to your system, click the Download More Languages button; doing so launches the Redistributable Downloader Wizard, which enables you to download one or more redistributables to your system.*

.NET 1.1/2.0 Language Packs Dialog Box



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Use the .NET 1.1/2.0 Language Packs dialog box to select the languages that correspond with the .NET language packs that you want to be installed on target systems. This dialog box opens when you click the ellipsis button (...) in the .NET 1.1/2.0 Language Packs setting on the .NET/J# tab for a release in the Releases view.

If you do not have a particular language pack installed on your system, the check box for that language is disabled in this dialog box.



***Tip** • To download one or more language packs to your system, click the Download More Languages button; doing so launches the Redistributable Downloader Wizard, which enables you to download language packs to your system.*

.NET Installer Class Arguments Dialog Box

The .NET Installer Class Arguments dialog box displays the arguments that you want to pass to the .NET Installer class. To launch the dialog box, click the ellipsis button (...) for the .NET Installer Class Arguments component setting. Click the Add Argument button beneath the argument that you want to add to launch the Add an Argument/Value Pair dialog box.

There are four methods on the Installer class that are called at various points during the installation: Install, Commit, Uninstall, and Rollback. You use this dialog to specify additional argument lists for the Installer class for each of the execution contexts.



Note *• Install custom actions are run when the component containing the Installer class custom action is installed. Uninstall custom actions are run when the component containing the Installer class custom action is uninstalled. These two types are run in the Execute sequence, right after StartServices and before RegisterUser.*

Using Arguments

For all methods, */LogFile=* is the default argument. This results in no log being generated when these methods are run. You can complete this argument by using Windows Installer properties in [brackets]. To add arguments, type in the appropriate edit field or use the Add Argument button.

Syntax Rules

When you click **OK**, InstallShield validates what is in the edit fields and displays a warning if the syntax is incorrect. The syntax rules include the following:

- All quotes must be matched.
- Escape characters are valid. If you use \" in your argument, it is not counted it when checking the quote matching.
- Every space-separated argument token must consist of a name preceded by a slash and followed by an equal sign (=). A value to the right of the equal sign is optional.
- Spaces inside quotes do not count when determining what is and is not part of an argument token. For example, in the argument */arg="hi there"*, *"hi there"* is all part of the argument token beginning with */arg*. In the argument */arg=hi there*, *there* is its own argument token and an invalid one.

Add an Argument/Value Pair Dialog Box

This dialog is displayed when you click one of the Add Argument buttons in the .NET Installer Class Arguments dialog box. When you are finished creating an argument, click OK to add the argument.

Dialog Options

Argument Name

Provide a name for the argument in the edit field.

Argument Value

Select one of the values from the option button list to create an argument. To create a custom argument, select the Custom option.

Property

Select a property from the menu to reference a property that exists in your installation project.

Directory

Use the drop-down menu to add a reference to a directory that exists in your installation project.

Custom

To create a custom argument, type an argument value in the edit field.

Add Existing Media Dialog Box

The Add Existing Media dialog box opens when you click the Add button in the Release Wizard's Update panel. This dialog box lets you select a release that is in the current project from a list of releases (excluding the current release).

Add Files for This Package Dialog Box



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Add Files for This Package dialog box opens when you add a package—for example, an .msi package or an .exe patch—in the Packages view of an Advanced UI or Suite/Advanced UI project.

This dialog box lets you specify whether the package file that you are adding requires other files and folders that are near the package that you are adding. For example, if you are adding an uncompressed .msi package to your project, you may need to include folders and other files that contain the files that the .msi package installs on target systems. These folders and other files are typically in the same folder that contains the .msi package.

The Add Files for This Package dialog box also lets you specify whether you want to use dynamic file linking for the additional files.

The options that are available on the Add Files for This Package dialog box are:

Table 12-1 • Add Files for This Package Dialog Box Settings

Setting	Description
Add Nothing	InstallShield includes only the package file in your project. If the package that you are adding to the project does not require any additional files, select this option. You may want to select this option if the package is compressed.

Table 12-1 • Add Files for This Package Dialog Box Settings (cont.)

Setting	Description
Add Adjacent Files	InstallShield includes the files that are in the same folder as the package file. InstallShield does not include any subfolders that are in the same folder as the package file, or any files that are in the subfolders.
Add Files in Subfolders	InstallShield includes the files that are in the subfolders of the folder that contains the package file that you are adding. InstallShield does not include any files that are in the same folder as the package file.
Add adjacent Files and Files in Subfolders	InstallShield includes the files that are in the same folder as the package file. InstallShield also includes the files that are in the subfolders of the folder that contains the package file that you are adding.
Add files dynamically found during build	To include the additional files as dynamic file links, select this check box. To include the additional files as static file links, clear this check box; by default, this check box is cleared. If the names of all of the additional files are known when you add the package to your project, you may want to use static file links for any additional files. If the additional files for your package may change over time, you may want to use dynamic file links for any additional files.  ***Note*** • *InstallShield always includes a static link for the package file (.msi, .msp, InstallScript package, or .exe). Therefore, the name of the package file should be consistent from one build to another.* *Selecting this check box has no effect if you click the Add Nothing option.* For more information about static and dynamic file linking, see Static vs. Dynamic Files for Packages in an Advanced UI or Suite/Advanced UI Project.

Add MIME Type Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Use the Add MIME Type dialog box to add or modify the mapping between a file extension and the program or interpreter that processes those files. This dialog box opens when you click Add or Edit on the MIME Types dialog box.

Table 12-2 • Add MIME Type Dialog Box Settings

Setting	Description
File Name Extension	Type the file name extension (for example, **.abc**). This is a static file name extension. To use a wild-card application mapping for an executable file, enter an asterisk (*).
MIME Type	Type the MIME type (for example, **application/octet-stream**).

Add New Method Dialog Box



Project • *This information applies to InstallScript Object projects.*

This dialog box lets you specify the name, return type, and parameter types of a new method and automatically paste corresponding code into your object script. This dialog box opens when you right-click the InstallScript view's Methods folder and select Add New Method.

Properties

Table 12-3 • Methods Properties

Property	Description
Name	Enter the name for your method.
Return Type	Select the data type that your method returns. You can choose from String, Number, Boolean, and Array.
Parameter List	Enter the parameter types that can be passed to your method.

Add New Property Dialog Box



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The Add New Property dialog box lets you specify the name and attributes of a new property and automatically paste corresponding code into your script. This dialog box opens when you right-click the InstallScript view's Properties folder and then click Add New Property.

Table 12-4 • Add New Property Dialog Box Settings

Setting	Description
Property Name	This is the global name of this property. Therefore, when trying to read from or write to this property from script, you will need to use this name. The name you enter in this box will be appended to the local variable name property.
Data Type	Select String, Number, or Boolean, depending on the type of data that your property requires. The local variable name will change depending on the selection that you make.
Access Method	Select the appropriate level of access for the property. Available options are Read/Write, Read Only, and Write Only.
Array	If your property is an array, select this check box.  ***Note*** • *If you plan on using the InstallShield stock wizard for your object's design-time interface, you cannot use array properties. You can create a custom wizard that accepts array properties or you can redesign the property without using an array.*
Local Variable Name	You can accept the default name or create your own variable name. This variable name is used when calling your variable from within the script.
Default Value	Enter the starting value for your property. Be sure to match the value to the data type that you chose. When you are specifying a string value, type it exactly as you would in the script; enclose the string value in quotation marks and use the appropriate escape sequences for special characters (for example, enter a backslash as \\). Do not enter a string identifier or a system variable in an InstallScript Object project; the initialization of the property's value is performed when the object is added to a project, at which time string identifiers and system variables are not yet initialized.

Add Predefined Search Dialog Box

The Add Predefined Search dialog box opens when you right-click the grid in the System Search view and then click Add predefined search.

Select a predefined value from the list in this dialog box and then click OK to add the search to the grid in the System Search view.

If you want system searches that have been published to a repository to be displayed in this dialog box, select the Show Searches in Repository check box.

Add to Source Control Dialog Box

This dialog is displayed when you add a project to your source control program through InstallShield.

Dialog Options

Comments

Enter the comments describing the file. These comments are stored in your source control program.

Keep checked out

Select this option to add the .ism file (in text format), but keep it checked out in your working directory.

Convert to XML

This option causes the .ism file to use the XML representation, which is better suited for source control. If you add a project in Binary format to source control, this option is available and is selected by default. It is recommended that you leave this option selected.



Note • *When a project file format is converted to either XML or binary format, the project file extension remains *.ism.*

Application Extension Mapping Dialog Box

Use the Application Extension Mapping dialog box to add or modify the mapping between a file extension and the program or interpreter that processes those files. This dialog box is launched when you click Add on the Application Mappings dialog box.

Table 12-5 • Application Extension Mapping Dialog Box Settings

Setting	Description
Extension	Type the file name extension that is associated with your application (for example, *.abc*). To use a wild-card application mapping for an executable file, enter an asterisk (*).
Executable	Type the path, or click Browse to launch the Select Executable dialog box. This enables you to specify the executable in your project to which you want to map. Type the name of the executable file (.exe or .dll) or use the Browse button to search for the file. The executable file must be located on your Web server's local hard drive.
Verbs	The Verbs section enables you to specify which HTTP verbs should be passed to the application. • **All verbs**—Select this option to include all verbs. This option passes all requests to an application. • **Limit to**—Select this option to specify the HTTP verbs that should be passed to an application. Separate verbs with a comma.

Table 12-5 • Application Extension Mapping Dialog Box Settings (cont.)

Setting	Description
Script Engine (IIS 6 and earlier only)	If you want the application to run in a directory without Execute permissions, select this check box. This setting is intended primarily for script-based applications, such as ASP and IDC, that are mapped to an interpreter. For a script-mapped application to run, either the Scripts Only or Scripts and Executables option must be selected for the Execute Permissions property. To allow only script-mapped applications to run, select the Scripts Only option. To allow both script-mapped applications and executable files (.exe and .dll) to run, select Scripts and Executables. This setting applies to IIS 6 and earlier. IIS 7 ignores this setting.
Check that file exists (IIS 6 and earlier only)	To instruct the Web server to verify the existence of the requested script file and to specify that the requesting user should have access permission for that script file, select this check box. If the script does not exist or the end user does not have permission, the appropriate warning message is returned to the browser and the script engine is not invoked. This option can be useful for scripts mapped to non-CGI executable files like the Perl interpreter that do not send a CGI response if the script is not accessible.  ***Note*** • *Because the script is opened twice—once by the server and once by the script engine—there is some performance cost when this check box is selected.* This setting applies to IIS 6 and earlier. IIS 7 ignores this setting.

Application Mappings Dialog Box

The Application Mappings dialog box lets you add, edit, and delete a mapping between a file name extension and the application that processes those files.

The Application Mappings dialog box is available from within the Internet Information Services view. To open this dialog box, click a Web site, application, or virtual directory in the explorer. Then click the ellipsis button (...) in the Application Mappings setting.



Note • *If an asterisk (*) appears in the Verbs column, all verbs will be used for the specified extension.*

Table 12-6 • Application Mappings Dialog Box Settings

Setting	Description
Add	To add an entry for mapping a file name extension and the program or interpreter that processes those files, click this button. This opens the Application Extension Mapping dialog box.

Table 12-6 • Application Mappings Dialog Box Settings (cont.)

Setting	Description
Edit	To edit an existing application mapping, select the mapping and click this button.
Remove	To delete an existing application mapping, select the extension and click this button.

Associate Components Dialog Box

The Associate Components dialog box enables you to associate components with features in the Setup Design view. Select the component or components that you want to associate with the current feature and click OK.

Note • *Components that are not associated with a feature are displayed with the orphaned component icon.*

For more information on component-feature relationships, see Associating New Components with Features.

Associate DIMs Dialog Box

The Associate DIMs dialog box enables you to associate DIMs with features in the Setup Design view. Select the DIM or DIMs that you want to associate with the current feature and click OK.

Note • *DIMs that are not associated with a feature are displayed with the orphaned component icon.*

For more information on DIM-feature relationships, see Associating a DIM Reference with a Feature.

Batch Build Dialog Box

This dialog displays a tree control containing all the product configurations and releases you have defined in the current project. Select the releases that you want to include in your batch build.

Dialog Options

Tree Control

Select the releases that you would like to build by checking the appropriate boxes. Selecting a product configuration icon selects all the releases associated with that product configuration.

Select All

Click this button to select all the releases in the tree.

Unselect All

Click this button to deselect all the releases in the tree.

Build

Click this button to build all the selected releases.

Browse for Directory Dialog Box

Use the Browse for Directory dialog box to browse to a directory, create a new directory, rename a directory, or delete a directory.

Dialog Box Options

Destination Directories

This field lists all of the currently available destination directories. You can select, create, rename, or delete directories in this field.

Selecting a Directory



Task ***To select a directory:***

1. Click on a directory to select it.
2. Click **OK**.

Creating a New Directory



Task ***To create a new directory:***

1. Select a directory or the Destination Computer and press the **Insert** key, or right-click and select **New** from the context menu. This creates a directory beneath the selected folder or beneath the Destination Computer.
2. Type the directory name.
3. Provide a directory identifier, if necessary.

Renaming a Directory



Task ***To rename a directory:***

1. Select a directory or the Destination Computer and press F2, or right-click and select **Rename** from the context menu.
2. Type the new directory name. Note that you cannot rename predefined directories.
3. Rename the directory identifier, if necessary, to be consistent with the directory's new name.

Deleting a Directory

Task ***To delete a directory:***

Select a directory and press the **Delete** key, or right-click and select **Delete** from the context menu.

Note that you cannot delete predefined directories.

Note *• When you delete a directory, any subdirectories beneath the selected directory are also deleted.*

Directory Identifier

You can use the **Directory Identifier** field to provide a user-friendly name for a directory. For example, if you have a directory—**ProgramFilesFolder\MyProgram\Graphics\Jpg**—you can use just JPG to identify the directory path. The **Component Destination** field in the IDE displays the path as {JPG} [ProgramFilesFolder]MyProgram\Graphics\Jpg.

Note *• The directory identifier must be a valid Windows Installer identifier. For features, the directory identifier must contain all capital letters.*

Browse for Directory/Shortcut Target Dialog Box

Use the Browse for Directory dialog box to browse to a directory, create a new directory, rename a directory, or delete a directory. Use the Browse for Shortcut Target dialog box to browse to a directory or file, create a new directory, rename a directory, delete a directory.

Dialog Box Options

Destination Directories

This field lists all of the currently available destination directories. You can select, create, rename, or delete directories in this field.

Selecting a Directory

Task ***To select a directory:***

1. Click on a directory to select it.
2. Click **OK.**

Creating a New Directory



Task ***To create a new directory:***

1. Select a directory or the Destination Computer and press the Insert key, or right-click and select **New** from the context menu. This creates a directory beneath the selected folder or beneath the Destination Computer.
2. Type the directory name.
3. Provide a directory identifier, if necessary.

Renaming a Directory



Task ***To rename a directory:***

1. Select a directory or the Destination Computer and press F2, or right-click and select **Rename** from the context menu.
2. Type the new directory name. Note that you cannot rename predefined directories.
3. Rename the directory identifier, if necessary, to be consistent with the directory's new name.

Deleting a Directory



Task ***To delete a directory:***

Select a directory and press the **Delete** key, or right-click and select **Delete** from the context menu. Note that you cannot delete predefined directories.



Note • *When you delete a directory, any subdirectories beneath the selected directory are also deleted.*

Hard-Coding a Shortcut



Task ***To hard code a shortcut to a path such as C:\Winnt\Notepad.exe:***

First create two folders — "C:" and "Winnt." Then enter "Notepad.exe" in the filename field of the **Browse for Directory** dialog or select the root node, and enter "C:\Winnt\Notepad.exe" in the filename field.

Directory Identifier

You can use the **Directory Identifier** field to provide a user-friendly name for a directory. For example, if you have a directory—**ProgramFilesFolder\MyProgram\Graphics\Jpg**—you can use just JPG to identify the directory path. The **Component Destination** field in the InstallShield interface displays the path as {JPG} [ProgramFilesFolder]MyProgram\Graphics\Jpg.

Note • *The directory identifier must be a valid MSI identifier. For features, the directory identifier must contain all uppercase letters.*

You can enter the file name in the **File name** edit box, which means that you are entering the selected directory\filename as the target of the shortcut.

Browse for File Dialog Box

The Browse for File dialog box is displayed when you click the ellipsis button (...) in the Include Patterns and Files setting or the Exclude Patterns and Files setting on the Signing tab for a release in the Releases view. The Browse for File dialog box lets you specify which static files in your project should or should not be signed. It also lets you use an asterisk (*) as a wild-card character. This is especially helpful if you include dynamically linked files in your project and you want to sign all files that match a certain pattern.

Table 12-7 • Settings on the Browse for File Dialog Box

Setting	Description
Select files to sign	This box is displayed if you click the ellipsis button (...) in the Include Patterns and Files setting. This box lists all of the statically included files in your project that match the file types that are selected in the **Show files of type** list. It also lists some default file patterns, such as *.dll. Select the check boxes for the files and file patterns that correspond with the types of files in your project that you want InstallShield to sign at build time.
Select files to skip signing	This box is displayed if you click the ellipsis button (...) in the Exclude Patterns and Files setting. This box lists all of the statically included files in your project that match the file types that are selected in the **Show files of type** list. It also lists some default file patterns, such as *.dll. If you want to prevent certain files or file patterns from being signed by InstallShield at build time, select the check boxes for the appropriate files and file patterns.
Show files of type	Use this list to filter the types of files that are displayed in the **Select files to sign** box or the **Select files to skip signing** box.

Windows Logo • *All executable files (including .exe, .dll, .ocx, .sys, .cpl, .drv, and .scr files) in an installation must be digitally signed for the Windows logo program.*

When you click OK on this dialog box, InstallShield adds new Include settings under the Include Patterns and Files setting or new Exclude settings under the Exclude Patterns and Files setting.

Note that the files and file patterns that should not be signed override any files and file patterns that should be signed. For example, if you specify *.exe in an Include setting and in an Exclude setting, InstallShield does not sign any .exe files.

Browse for File to Run Dialog Box

The Browse for File to Run dialog box opens when you click the ellipsis button (...) in the Target setting in the Scheduled Task view. This dialog box lets you specify the file name and location on the target system for the file that you want the scheduled task to run.

Table 12-8 • Settings on the Browse for File to Run Dialog Box

Setting	Description
Destination Directories	Select the directory on the target system that will contain the file that you want to be run as a scheduled task. To specify a new directory, right-click the location that you want to contain the new directory, and then click New Directory. To hide or show empty predefined directories in this box, right-click in this box and then click Show Empty Predefined Directories.
Directory Identifier	If you select a destination directory that you created in the Destination Directories box, you can use this setting to enter a user-friendly name for the directory. For example, if you have a directory such as **[ProgramFilesFolder]\MyProgram\Graphics\JPG**, you can use just JPG to identify the directory path. The Component Destination field in InstallShield displays the path as **{JPG} [ProgramFilesFolder]MyProgram\Graphics\JPG**. If you select a predefined directory in the Destination Directories box, this setting is read-only.
File Name	Enter the name of the file that you want the scheduled task to launch.

Browse for Shortcut Icon Dialog Box



Project • *This applies to InstallScript projects.*

Use the Browse for Shortcut Icon dialog box to specify to the location on the target system of the icon file (.ico) or the .exe or .dll file that contains the icon resource. This dialog box opens when you click the ellipsis button (...) in the Icon setting in the Shortcuts view.

Table 12-9 • Browse for Shortcut Icon Dialog Box Settings

Setting	Description
Destination Directories	Select the directory on the target system that contains the icon file.

Table 12-9 • Browse for Shortcut Icon Dialog Box Settings (cont.)

Setting	Description
Current Value	This read-only setting shows the selected directory that contains the icon file.
File Name	Enter the name of the icon file (.ico) or the .exe or .dll file that contains the icon resource.



Tip • *To use a hard-coded path for a file, enter the path in the Icon File setting manually, instead of using the Browse for Shortcut Icon dialog box to use system variables for directories. The Icon File setting is under the Icon setting in the Shortcuts view.*

Browse for Tile Target Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

InstallShield supports configuring the appearance of a desktop app's tile on the Start screen. Use the Browse for Tile Target dialog box to navigate to and select the .exe file for the app whose tile appearance you want to configure.

Table 12-10 • Browse for Tile Target Dialog Box Settings

Setting	Description
Look in	Browse to the directory that contains the .exe file of the app whose tile appearance you want to configure. Selecting a folder from this list displays the contents of the folder in the primary pane. In the primary pane, click an .exe file to select it.
File Name	After you have selected an .exe file in the primary pane, its file name is displayed here.

Certificate Selection Dialog Box



Edition • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*
- *QuickPatch*

When you are configuring digital signature information for a release, a patch, or a QuickPatch package, use the Certificate Selection dialog box to specify which certificate you want to use to sign your files. InstallShield lets you choose between the following options:

- You can specify the .pfx certificate file on your machine that you want to use for signing.
- You can reference a certificate store that contains the certificate that you want to use for signing.

Accessing the Certificate Selection Dialog Box

Instructions on how to access the Certificate Selection dialog box depend on whether you are specifying certificate information for a release, a patch, or a QuickPatch package.



Task ***To access the Certificate Selection dialog box for a release:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the release that you want to configure.
3. Click the **Signing** tab.
4. In the **Digital Certificate Information** setting, click the ellipsis button (**...**).



Task ***To access the Certificate Selection dialog box for a patch:***

1. In the View List under **Media**, click **Patch Design**.
2. In the **Patch Design** explorer, select the patch configure that you want to modify.
3. Click the **Digital Signature** tab.
4. Next to the **Digital Certificate Information** setting, click the **Browse** button.



Task ***To access the Certificate Selection dialog box in a QuickPatch project:***

1. In the View List under **Patch Settings**, click **General Information**.
2. In the **General Information** explorer, click **Build Settings**.
3. Click the **Digital Signature** tab.
4. Next to the **Digital Certificate Information** setting, click the **Browse** button.

Certificate Selection Dialog Box Settings

Table 12-11 • Certificate Selection Dialog Box Settings

Setting	Description
Use a file (.pfx)	To use a .pfx file to digitally sign your release at build time, select this option. Then specify the location of your .pfx. You can type the path to the file or use the ellipsis button (...) to browse to the file location.
Use a certificate store	To reference a certificate store that contains the certificate that you want to use to digitally sign your release at build time, select this option and then enter values in the subsettings under this option.
Certificate Store Name	Select the name of the certificate store that contains the certificate that you want to use. Available options are: • Personal • Trusted Root Certification Authorities • Enterprise Trust • Intermediate Certification Authorities This setting is available if you select the **Use a certificate store** option.
Certificate Store Location	Select the location of the certificate store that contains the certificate that you want to use. Available options are: • User • Machine This setting is available if you select the **Use a certificate store** option.
Certificate Subject	Enter the subject of the certificate that you want to use, or select from the list of certificates that are available on your machine. This setting is available if you select the **Use a certificate store** option.
Signature Digest	Choose the signature digest hashing algorithm (or choose to let InstallShield specify it automatically based on the certificate hash). Available options are: • Based on certificate hash • SHA-1 • SHA-256

Check In Dialog Box

When you check an InstallShield project in to your source code control application through InstallShield, the Check

In dialog box opens.

Table 12-12 • Check In Dialog Box Settings

Setting	Description
Comments	Enter comments that describe the changes that you made to the file. These comments are stored in your source code control application.
Keep checked out	To check in the changed version of the .isv file, but keep it checked out in your working directory, select this option.
Differences	To view a line-by-line comparison between the .ism file (in text format) that you are checking in with the previous version of the file, click this button.

Check Out Dialog Box

If you select the **Use dialog for checkout** check box on the Source Control tab of the Options dialog box, the Check Out dialog box opens when you check out an InstallShield project from your source code control application through InstallShield.

Table 12-13 • Check Out Dialog Box Options

Option	Description
Comments	Type comments about your reasons for checking out the .ism file.
Advanced	To configure all of your source code control application's settings for checking out the project file, click this button.

Component Properties Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Component Properties dialog box lets you set some component properties—including dynamic file link settings and associated features—from the Files and Folders view.

Task ***To launch the dialog box:***

1. In the View List under **Application Data**, click **Files and Folders**.
2. Ensure that components are displayed in the **Destination computer's folders** pane.

   If components are not displayed: In the **Destination computer's folders** pane, right-click **Destination Computer** and click **Show Components** (in Windows Installer–based projects) or **Show Components and Subfolders** (in InstallScript projects).
3. Right-click a component and then click **Properties**. The **Component Properties** dialog box opens.

The following tabs are available on this dialog box:

- General tab
- File Linking tab
- Features tab

General Tab

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The General tab on the Component Properties dialog box is where you set the properties for the selected component.

Table 12-14 • Settings on the General Tab of the Component Properties Dialog Box

Setting	Project Type	Description
Shared	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, MSI Database, MSM Database, Transform	If the files that are contained within this component are shared files, select this check box. To learn more, see Managing Reference Counts for Shared Files.

Table 12-14 • Settings on the General Tab of the Component Properties Dialog Box (cont.)

Setting	Project Type	Description
Never Overwrite	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	If you do not want to replace an existing version of the same file that is already present on the target machine, select this check box. To learn more, see Checking File Versions Before Installing.
Permanent	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	If the files in this component should not be removed from the system during uninstallation, select this check box. For example, validation rule ICE09 requires any component with a destination of [SystemFolder] to be marked as Permanent. To learn more, see Specifying Whether a Component's Files and Other Associated Data Are Uninstalled During Uninstallation.
Uninstall	InstallScript, InstallScript Object	If the files in this component should be removed when the application is uninstalled, select this check box. To learn more, see Specifying Whether a Component's Files and Other Associated Data Are Uninstalled During Uninstallation.
64-bit	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	To mark this component as 64 bit, select this check box. If a 64-bit component is included in your installation, the installation cannot be run on 32-bit machines.
Extract COM Info At Build	Basic MSI, DIM, InstallScript MSI, Merge Module	If you want InstallShield to extract the COM information each time you build a release, select this check box. If your COM interfaces are subject to change, this option is preferable to maintaining statically acquired or provided data in the COM Registration advanced setting. If you have marked the component's COM server file as Self-Registering, you should not select this check box.

Table 12-14 • Settings on the General Tab of the Component Properties Dialog Box (cont.)

Setting	Project Type	Description
Self-register	InstallScript, InstallScript Object	If the component's files are self-registering, select this check box. A self-registering file is an OLE server that can place information about itself in the registry so that it is seen by other OLE applications. Such a file can also remove this information from the registry when it is uninstalled. ***Project*** • *InstallScript and InstallScript Object projects support self-registration of .dll files, .exe files, and type libraries; that is, .tlb and .olb files. In Windows Installer–based projects, self-registration is set at the file level, rather than the component level.*

File Linking Tab

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The File Linking tab on the Component Properties dialog box shows all of the settings for your dynamic link.

Table 12-15 • Settings on the File Linking Tab of the Component Properties Dialog Box

Setting	Description
Source	This column shows where the source files are stored. A path variable is typically used.
Include Subfolders	This column indicates whether InstallShield should dynamically link the files in each subfolder. For information on how InstallShield creates components for the dynamically linked files in subfolders, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.
Self-Register	This column indicates whether you want InstallShield to self-register every file in the dynamic link. If the file are part of a 64-bit component and the Self-Register column indicates Yes for a dynamic link, the installation performs 64-bit self-registration on the target system. For more information, see Targeting 64-Bit Operating Systems.

Table 12-15 • Settings on the File Linking Tab of the Component Properties Dialog Box (cont.)

Setting	Description
Create Components	This column indicates how you want InstallShield to create the components when it adds the dynamic files to your release at build time. The valid options are: • **Best Practice**—InstallShield adheres to best practices when creating components for the dynamically linked files. • **By Directory**—InstallShield creates one component for each folder and subfolder. To learn about these two methods, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.
Details	This column indicates the filter criteria for including files in and excluding files from the dynamic link.



Tip • *Note the following guidelines for working with the File Linking tab:*

- *To add a new dynamic link, click the New Link button. InstallShield displays the Dynamic File Link Settings dialog box.*
- *To modify the settings for an existing link, select the link and then click the Modify button. InstallShield displays the Dynamic File Link Settings dialog box.*
- *To remove a link and its associated dynamically linked files, select the link and then click the Delete button.*

Features Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript Object*
- *MSI Database*
- *Transform*

The Features tab on the Component Properties dialog box is where you specify the features with which the component is associated. Select the check box next to the feature or features with which you want the component to be associated.

Condition Builder Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Condition Builder dialog box simplifies the process of creating a condition for a component, a dialog, a custom action, a SQL script, a system search, or other installation element by providing a selection of valid Windows Installer properties and conditional expression operators.

The Condition Builder dialog box contains the following settings.

Table 12-16 • Condition Builder Dialog Box Settings

Setting	Description
Conditions	This box is where you define the condition for your component, dialog, custom action, or other installation element. You can type a condition directly in this box, or you can use the Properties list, the Operators list, and the Add buttons to build a condition statement.  ***Important*** • *When you click the OK button on this dialog box, InstallShield performs basic condition validation; however, you should still double-check that your condition statements evaluate to the expected outcome. For more information and example conditions, see Building Conditional Statements.*
Properties	This list contains common Windows Installer properties, as well as any properties that have been added in the Property Manager view. Select the property that you want to evaluate, and then click the **Add** button next to this list. If the list does not contain the property that you want to evaluate, you can type the appropriate property name in the Conditions box.
Operators	This list contains all of the valid operators that are recognized by the Windows Installer. If your conditional statement should contain an operator, select the operator that you want to use and then click the **Add** button. InstallShield adds the operator to the conditional statement. If you add an operator, type the appropriate value after the operator.

Content Source Path Dialog Box

The Content Source Path dialog box opens when you click the UNC button in the Content Source Path (Local or UNC) setting for a Web site in the Internet Information Services view. This dialog box contains the following setting.

Table 12-17 • Content Source Path Dialog Box Setting

Setting	Description
Universal Naming Convention content source path	Specify the UNC path for the IIS Web site's files. For example: **\\server\share**

Convert InstallScript MSI Dialog Box

The Convert InstallScript MSI dialog box lets you convert an InstallScript MSI project to an InstallScript project. An InstallScript project offers the advantages of InstallScript MSI—such as dialog box skins, a flexible end-user interface, and a powerful scripting model—without any of the limitations or overhead of Windows Installer.

This dialog box is displayed when you select the Project menu's Convert to InstallScript Project command.

If you click Yes, most project elements are converted, many with no change. The following changes take place (the conversion process issues many messages to let you know what is happening):

- INSTALLDIR is changed to TARGETDIR in the following IDE elements:
  - Component destinations
  - Path specifications in the Files and Folders view
  - Target specifications in the Shortcuts view

    Note that the value of TARGETDIR must be set in your script, as it is in the following default code in the OnFirstUIBefore event handler function:

    ```
    if ( ALLUSERS ) then
        TARGETDIR = PROGRAMFILES ^ IFX_COMPANY_NAME ^ IFX_PRODUCT_NAME;
    else
        TARGETDIR = FOLDER_APPDATA ^ IFX_COMPANY_NAME ^ IFX_PRODUCT_NAME;
    endif;
    ```
- A new product code is generated.
- A registry set is created for each component that had registry data associated with it.
- Each component with multiple dynamic file links is split into multiple components with one dynamic file link in each.
- The script file Setup.rul is renamed OldSetup.rul and is displayed in the InstallScript view. Note that no changes are made to this script file.
- Warnings are issued for the following unsupported items:
  - Unsupported folder specifiers in path specifications in the Files and Folders view, or destinations or target specifications in the Shortcuts view
  - Win32 assemblies that are associated with components

- Global assembly caches that are associated with components
- Registry value names that contain a hyphen (-) or asterisk (*)
- Advertised shortcuts
- Folders under the Support Files view's Disk1 folder
- Merge modules (you can re-include these in the converted project from the Objects view)
- Unsupported MSI tables that contain data

Releases and your scripts are not converted. At a minimum, you need to modify your scripts to set TARGETDIR, as mentioned above, change all INSTALLDIR references to TARGETDIR, and remove calls to Windows Installer API functions (whose names begin with Msi). Furthermore, the obsolete string entry PRODUCT_NAME is not converted; IFX_PRODUCT_NAME can be used instead.

Note that setting SHELL_OBJECT_FOLDER in OnFirstUIBefore is not necessary and the default script line SHELL_OBJECT_FOLDER = @PRODUCT_NAME; should be deleted.

The following Windows Installer folder specifiers are not supported in InstallScript projects:

- ALLUSERSPROFILE
- AdminToolsFolder
- AppDataFolder
- CommonAppDataFolder
- CommonFiles64Folder
- FavoritesFolder
- FontsFolder
- GlobalAssemblyCache
- LocalAppDataFolder
- MyPicturesFolder
- PersonalFolder
- PrimaryVolumePath
- ProgramFiles64Folder
- SendToFolder
- System16Folder
- System64Folder
- TempFolder
- TemplateFolder
- USERPROFILE
- WindowsVolume

In addition, the following folder specifiers are not supported for shortcut destinations:

- ProgramFilesFolder
- CommonFilesFolder
- WindowsFolder
- SystemFolder

Create a New Component Dialog Box

Some aspects of your installation project must be associated with a component. For example, if no component exists when a shortcut or an .ini file change is created, InstallShield prompts you to create a new component.

Dialog Options

New component name

InstallShield provides a default name for the component. Type a meaningful name into this field. The name you provide is for your reference only and is never displayed to end users.

Select the features to associate this component with



Project • *This property applies to the following project types:*

- *Basic MSI*
- *InstallScript*

Select from the list the feature or features with which you want to associate the new component. If no features appear in this field, you can add one by clicking New Feature. You are not required to associate your new component with a feature; however, if you do not, this component will not be installed.

New Feature



Project • *This property applies to the following project types:*

- *Basic MSI*
- *InstallScript*

Click New Feature to add a feature to your setup project. When you have created a feature, you can associate your new component with it by clicking the box next to the feature name.

Do not show this dialog again

Select this option if you want InstallShield to automatically create any necessary components in the future.

Create a New Feature Dialog Box

Some aspects of your installation project must be associated with a feature. For example, if no feature exists when a merge module is added, or an ODBC resource is configured, InstallShield prompts you to create a feature.

Dialog Options

New feature name

InstallShield creates a default feature name for you. Provide a meaningful name for the feature. The name you enter here is for your reference only and is never displayed to end users.

Do not show this dialog again

Select this check box if you want InstallShield to automatically create components and features when required.

Custom Error Handling Dialog Box

Specify how to handle specific SQL errors in the Custom Error Handling dialog box. Any error you define will override the script's default error handling. You can define how to handle an error for all scripts in your project or simply for this script.

Error Number

Enter the SQL error number for which you want to customize error handling.

Behavior

Override the default behavior by choosing one of the following options:

- On Error, Abort Installation
- On Error, Go to Next Script
- On Error, Go to Next Statement

Project Wide

To apply the customized error handling to all scripts in your project, select Yes. To apply it to only this script, select No.

Custom Errors Dialog Box

The Custom Errors dialog box lists all of the HTTP errors that can be customized for a Web site, application, or virtual directory. A custom error can be a URL or a pointer to a file on the server.



Task ***To configure error messages:***

1. In the **Custom Errors** dialog box, select one or more errors.
2. Click **Edit**. The **Error Mapping Properties** dialog box opens.
3. Select a message type and specify a file or URL as appropriate.
4. Click **OK**.

You can click the Set to Default button to restore default settings for the selected error.

Customize Validation Settings Dialog Box

The Customize Validation Settings dialog box lets you specify which internal consistency evaluators (ICEs) should be used for a specific validation suite. This dialog box opens when you click Customize on the Validation tab of the Options dialog box.

To customize the list of ICEs that are performed during a particular type of validation, select the validation suite in the CUB file list. Then select the check boxes for each of the ICEs that should be performed, and clear the check boxes for each of the ICEs that should be ignored.

To learn about specific ICEs, see ICE Reference in the Windows Installer Help Library.

For information about each of the InstallShield ICEs that were added to the InstallShield validation suites, see ISICEs.

Data Languages Dialog Box



Edition • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

The Data Languages dialog box lets you include certain components and exclude others based on the language that is selected for each component. If the language specified for a component does not match one of the languages that is selected in this dialog box, InstallShield does not include the component in the release.

To open the Data Languages dialog box, open the Releases view and select a release. Then click the ellipsis button (...) in the Data Languages setting on the Build tab.

By default, releases are language independent; that is, none of the project's components are excluded from the release.

Delete Property Dialog Box

The Delete Property dialog box is displayed when you delete a system search associated with a property in your installation project. The goal of the dialog box is to determine whether or not you want to delete that property along with the search.



Note • *Deleting the property may affect other areas of your setup.*

Select Yes if you are certain that your project will not be affected in the absence of that property. Select No if you are uncertain whether it will affect your installation.

Dependencies Dialog Box

The Dependencies dialog box displays the list of dependencies for a selected file. To access this dialog box, right-click a file in the **Destination computer's files** pane of the Files and Folders view, and then click **Dependencies from scan at build**.

The dialog box provides results for assembly DLLs. If you launch the dialog box from Microsoft Visual Studio, the dialog box shows results for project outputs. If launched from InstallShield outside of Visual Studio, scan at build dependencies is disabled for project outputs.



Note • *This is enabled only for key files and when the .NET Scan at Build property is set to Dependencies and Properties. If the dependency or one of its dependencies cannot be located, a red icon is displayed.*

Dialog Box Options

Dependency

This section lists all of the dependencies with a check box next to each. To exclude a dependency from the build, clear the check box next to the dependency. To close the dialog, click OK.



Note • *Any new dependencies detected at build time—for files added after closing the Dependencies dialog box—are added to the build.*

Dialog Images Dialog Box

Use the Dialog Images dialog box to add an image (.bmp, .gif, .jpg, .jpeg, or .ibd) that you want to display on your installation's dialogs.

Full Screen Image

Browse to a graphic file that will serve as the full-screen background for your *exterior* dialogs. Exterior dialogs are dialogs that appear at the beginning or end of the installation, including InstallWelcome and SetupCompleteSuccess (the final dialog upon successful installation). The full-screen image should measure 499 by 312 pixels.

Banner Image

Browse to a graphic file that will run across the top of interior dialogs. Interior dialogs, which appear between the first and last installation dialogs, include the LicenseAgreement and CustomSetup dialogs. The banner image should be 499 by 58 pixels.

Digitally Sign Setup Dialog Box

The Digitally Sign Setup dialog box is displayed when you click Digitally Sign Setup on the Build Installation page of the Project Assistant. This dialog box enables you to assure your end users that the code within your application has not been modified or corrupted since publication.

Table 12-18 • Digitally Sign Setup Dialog Box Settings

Setting	Description
Digitally sign setup	To digitally sign your installation, select this check box The other settings on this dialog box are enabled when you select this check box.
Certificate URL	Type a fully qualified URL—for example, **http://www.mydomain.com**. This URL is used in your digital signature to link to a site that you would like end users to visit to learn more about your product, organization, or company.
Digital certificate information	To specify the digital certificate that you want to use to sign your release, click the ellipsis button (...) next to this setting. The Certificate Selection dialog box opens, enabling you to specify either the location of the .pfx file or information about the certificate store that contains the certificate. To learn more, see Certificate Selection Dialog Box.
Password	If the .pfx file that you are using has a password, enter it. InstallShield encrypts the password and stores it in your project file (.ism). At build time, InstallShield uses the password to sign files with a .pfx file. If your certificate is protected by a password but you do not enter it in this setting, signing with a .pfx file fails. Note that if you configure your project to use a certificate that was imported with password protection into a store, Windows prompts for the password at build time when InstallShield is attempting to sign your project's files. The strong key protection that Windows uses does not permit InstallShield to provide the password to the cryptographic provider.

Tip • *The Signing tab in the Releases view lets you specify which portions of your installation should be digitally signed at build time. InstallShield enables you to sign any and all of the following files in a release, depending on what type of project you are using:*

- *Windows Installer package (.msi file) for Basic MSI and InstallScript MSI projects*
- *Merge module package (.msm file) for Merge Module projects*
- **Setup.exe** *file for Basic MSI and InstallScript MSI projects*
- *Media header file for InstallScript projects*
- *Package (self-extracting single-executable file) for InstallScript projects*
- *Any files in your release, including your application files*

To learn more, see Digitally Signing a Release and Its Files at Build Time.



Windows Logo • *All executable files (including .exe, .dll, .ocx, .sys, .cpl, .drv, and .scr files) in an installation must be digitally signed for the Windows logo program.*

Differences Dialog Box

The Differences dialog box is displayed when you click the Show Differences command, which is available from the Project menu's Source Control menu. The dialog box has a list that contains all of the Windows Installer tables in your installation project.



Task ***To view the table's differences in your source control program:***

Select a table from the list and click **OK**.

Dynamic File Link Settings Dialog Box

The Dynamic File Link Settings dialog box enables you to specify the source folder for your dynamic link. You can then choose to include all the files in that folder or only specified file types.

The Dynamic Link Settings dialog box is available from the File Linking tab of the Component Properties dialog box, as well as the Modify Dynamic Links dialog box.

Table 12-19 • Settings on the Dynamic File Link Settings Dialog Box

Setting	Description
Source folder	Enter the full path to the folder that you want to have dynamically linked, or click the Browse button to navigate to it. You can identify the source folder with any of the following methods: • Enter an absolute path to the folder, such as **C:\Build\MySourceFolder\Bitmaps**. • Enter a path variable. Follow the path variable with a backslash to specify a subfolder—for example, **<CommonFilesFolder>\InstallShield\IScript**. • Click the Browse button to navigate to a folder.
Include subfolders	To dynamically link the files in each subfolder, select this check box. For information on how InstallShield creates components for the dynamically linked files in subfolders, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.
Self-register all files	To self-register every file in the dynamic link, select this check box. If the files are part of a 64-bit component and you select this check box, the installation performs 64-bit self-registration on the target system. For more information, see Targeting 64-Bit Operating Systems.

Table 12-19 • Settings on the Dynamic File Link Settings Dialog Box (cont.)

Setting	Description
Create best practice components	To specify that InstallShield should adhere to best practices when creating the components for dynamically linked files, select this check box. When best practices for component creation are followed, InstallShield performs the following tasks at build time for all of the files that meet the include and exclude filter criteria: • InstallShield creates a separate component for each portable executable (PE) file in the dynamically linked folder. Each PE file is the key file of its component. • InstallShield adds all non-PE files at the root level of the dynamic link to the component that contains the link. • If the dynamic link includes a subfolder, InstallShield creates a new component for all of the non-PE files in that subfolder. If the dynamic link includes more than one subfolder, InstallShield creates a separate component for all of the non-PE files in each subfolder. To specify that InstallShield should not follow best practices when creating the components for dynamically linked files, clear this check box. For this component creation method, InstallShield performs the following tasks at build time for all of the files that meet the include and exclude filter criteria: • InstallShield creates one component for all of the files that are in the root-level dynamically linked folder, regardless of the file types. • If the dynamic link includes one or more subfolders, InstallShield creates a separate component for all of the files in each subfolder, regardless of the file types. The first dynamically linked file in a subfolder's component is the key file of that component. This check box is selected by default for all new dynamic links. For more information, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.
Include all files	To include the entire contents of your linked directory in your installation, select this option.

Table 12-19 • Settings on the Dynamic File Link Settings Dialog Box (cont.)

Setting	Description
Include/exclude files based on the following wild-card patterns	To include or exclude file types, select this option. Enter the file extension in the include or exclude field, preceded by an asterisk (*). Separate multiple entries with a comma. For example, if all of your image files are in one folder along with sound files and you want to dynamically link only the image files, you could specify that you would like to include only .bmp and .ico files in the dynamically linked folder. To do so, you would use an asterisk (*) in your include pattern, as in the following example: ***.bmp, *.ico** To include or exclude a specific file, you would enter the full file name in the include or exclude pattern box.

Edit Data Dialog Box

The Edit Data dialog is available from within the Application Registry panel in the Project Assistant. To access the dialog, double-click on a registry value in the Destination computer's registry data pane. The Multi-Line String Value dialog box is displayed for multi-string values.

Dialog Options

Value name

The name of the registry value as it appears in the Destination computer's registry data pane. To change the value name, right-click on the value and select Rename.

Value data

Contains the information that should be stored in the registry value. To change or edit the data, type in the edit field.

Edit IIS Metabase Value Dialog Box

InstallShield displays the Edit IIS Metabase Value dialog box if you click the Change Value button on the Other IIS Properties dialog box. The Other IIS Properties dialog box is displayed if you click the ellipsis button (...) in the Other IIS Properties setting for a selected Web site, application, or virtual directory in the Internet Information Services view.

The Edit IIS Metabase Value dialog box is where you specify the value for an IIS setting that is not displayed in the other areas of the Internet Information Services view.

If you change the default value for any advanced property, InstallShield adds the property, the value that you specify, and the additional required information to the **ISIISProperty** table.

Table 12-20 • Edit IIS Metabase Value Dialog Box Settings

Setting	Description
Property Name	This read-only setting shows the property that you selected to configure.
Value	Specify the value that you want to use for the selected IIS property.

Edit List Options Dialog Box

The Edit List Options dialog box



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Use the Edit List Options dialog box to define the list of options that you want to display in a combo box control or the list box control of your installation's user interface. To learn more, see Populating Combo Box and List Box Controls in the Wizard Interface.



Task ***To access the Edit List Options dialog box:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, expand the wizard page or secondary window that contains the combo box control or list box control that you want to configure.
3. In the **Content Property** setting, click the ellipsis button (**...**).

Table 12-21 • Settings on the Edit List Options Dialog Box

Setting	Description
New	To add a new row to the table, click this button.
Delete	To remove one or more selected rows, click this button.

Table 12-21 • Settings on the Edit List Options Dialog Box (cont.)

Setting	Description
Up	To move one or more selected rows up, click this button. The installation displays the list options in the order that you list them on this dialog box. To select multiple consecutive rows, click the first row, press and hold Shift, and then click the last row. To select multiple non-consecutive rows, click one of the rows, press and hold down CTRL, and then click each additional row.
Down	To move one or more selected rows down, click this button. The installation displays the list options in the order that you list them on this dialog box. To select multiple consecutive rows, click the first row, press and hold Shift, and then click the last row. To select multiple non-consecutive rows, click one of the rows, press and hold down CTRL, and then click each additional row.
Text	This column contains the list of options that you want to display in the control. When you type a value for this column, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Value	Each row in this column contains a value that corresponds with a list option for the selected wizard interface control. This is the property value that is stored in the property that is specified in the Property setting for the selected control.

Edit Registry Data Dialog Box

The Edit Registry dialog box enables you to edit the registry data within your project. To launch this dialog box, right-click a value in the Registry view and click Modify.

Table 12-22 • Edit Registry Data Dialog Box Settings

Setting	Description
Value name	This read-only setting shows the name of the registry value. To change the name of a value, close the Edit Registry Data dialog box, select the value that you want to rename, press F2, and type the new name.
Value data	Enter the data for this registry value as you would like it to appear on the target system.  ***Tip*** • *To write the value of a Windows Installer property called PropertyName to the registry, use the expression "[PropertyName]" in the value data.*

Edit Virtual Machine Configurations Dialog Box



***Edition** • Support for distributing releases to virtual machines is available in the Premier edition of InstallShield.*



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

Use the Edit Virtual Machine Configurations dialog box to configure virtual machine (VM) settings that you want to use when distributing successfully built releases to VMs at build time or on demand.



***Tip** • When you specify VM configuration settings in the Edit Virtual Machine Configurations dialog box, InstallShield writes the data that you specify to a file called* **VMConfigurations.xml**. *This* **VMConfigurations.xml** *file is a machine-wide file that you can configure once and then use it in other InstallShield projects, as well as share it with other team members. You can also use this file on your build machine. To learn more, see Sharing Virtual Machine Settings for Distribution of Releases.*

When you configure VM details in the subsettings under the Configuration setting on the Events tab in the Releases view, you overriding some of the machine-wide values that were entered on the Edit Virtual Machine Configurations dialog box.



Task ***To access the Edit Virtual Machine Configurations dialog box:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, select the release that you want to configure.
3. Click the **Events** tab, and then find the **Virtual Machine** area.
4. In the **Configuration** setting, click the ellipsis button (**...**).

The Edit Virtual Machine Configurations dialog box opens. The left part of this dialog box is where you manage the list of VM configurations that you want to available for selection. To add a new configuration, click the New button. To delete an existing configuration, select it and then click the Delete button.

When you select a specific configuration in the Configurations box, you can configure its settings on the right. The settings are organized into the following main categories:

- General
- Server
- Virtual Machine

General Settings

Use the General area to specify the name of the selected VM configuration.

Table 12-23 • General Settings on the Edit Virtual Machine Configurations Dialog Box

Setting	Description
Name	Enter a name for the group of VM configuration settings that you want to use when distributing the selected release to a VM.

Server Settings

Use the Server area to specify details about the server that hosts the VM to which you want to distribute releases.

Table 12-24 • Server Settings on the Edit Virtual Machine Configurations Dialog Box

Setting	Description
Server Type	Select the type of server that hosts the VM to which you want to distribute releases. Available options are: • Microsoft Hyper-V Server • VMware ESX or ESXi Server • VMware Workstation
Server Name	This setting is available for the following server types: Microsoft Hyper-V Server and VMware ESX or ESXi Server. Specify the name of the server that hosts the VM to which you want to distribute releases.
Authentication	This setting is available for the following server types: Microsoft Hyper-V Server and VMware ESX or ESXi Server. Select the type of authentication that you want to use to connect to the server that hosts the VM to which you want to distribute releases.
User Name	This setting is available for the following server types: Microsoft Hyper-V Server and VMware ESX or ESXi Server; it is applicable to server authentication. Enter the user name for the account that you want to use to connect to the server that hosts the VM to which you want to distribute releases.
Password	This setting is available for the following server types: Microsoft Hyper-V Server and VMware ESX or ESXi Server; it is applicable to server authentication. Enter the password for the account that you want to use to connect to the server that hosts the VM to which you want to distribute releases.

Virtual Machine Settings

Use the Virtual Machine area to specify details about the server that hosts the VM to which you want to distribute releases.

Table 12-25 • Virtual Machine Settings on the Edit Virtual Machine Configurations Dialog Box

Setting	Description
Machine Name	Specify the name of the VM to which you want to distribute releases.
Snapshot	Optionally specify the name of the snapshot that you want to use for distributing releases. Note that you can override the snapshot for an individual release. If no snapshot is specified for the VM configuration, InstallShield does not revert to a specific snapshot. InstallShield powers on the VM without reverting it to a specific snapshot, and copies your installation to the VM.
Authentication	Optionally select the type of authentication that you want to use to connect to the VM to which you want to distribute releases. Note that you can override the authentication for an individual release.
User Name	This setting is applicable to server authentication. Optionally enter the user name for the account that you want to use to connect to the VM to which you want to distribute releases. Note that you can override the user name for an individual release.
Password	This setting is applicable to server authentication. Optionally enter the password for the account that you want to use to connect to the VM to which you want to distribute releases. Note that you can override the password for an individual release.
Destination Path	Optionally specify the path on the VM to which you want InstallShield to stage the built release. The last subfolder in the path can be a path that InstallShield creates on the VM; the other folders in the path must already exist. Note that you can override the destination path for an individual release.

Error Mapping Properties Dialog Box

The Error Mapping Properties dialog box displays an IIS error code and its default properties. Use the Message Type list to set the message to a URL or a pointer to a file on the server. When you set the error message to a URL, you must specify the full URL. When you set the error message to a file pointer, you can browse for the file that you want to use as the error message or you can type the full path of the file to be used. This can include files that are not in your project but are pre-existing on the system.

Exclusion Languages Dialog Box



Project • *This information applies to the following project types:*

- *Merge Module*
- *MSM Database*

The Exclusion Languages dialog box opens when you click the ellipsis button (...) in the Language setting for a merge module exclusion in the General Information view. This dialog box lets you specify the language requirements for the merge module that is not compatible with the merge module that you are creating.

To specify language requirements for the merge module exclusion, do one of the following:

- To exclude a merge module if its language is a particular language, select the language in the language list, and then select the **matches** option.
- To exclude a merge module if its language is not a particular language, select the language in the language list, and then select the **does not match** option.
- If the language should not be used to exclude a merge module, select Language Independent in the language list.

Export Tables Dialog Box

The Export Tables dialog box provides a way to export one or more tables through the Direct Editor as .idt files.

Table 12-26 • Export Tables Dialog Box Settings

Setting	Description
Output Directory	Specify the directory where the tables are to be exported. You can also click Browse to navigate to the directory.
Tables	This box lists check boxes for all of the tables in your project. Select the check boxes of the tables that you want to export.
Select All	Selects all tables for export.
Clear All	Deselects all tables.
Invert	Simultaneously deselects any selected tables and selects any deselected tables.

Feature Condition Builder Dialog Box

The Feature Condition Builder dialog box simplifies the process of creating feature conditions by providing a selection of valid Windows Installer properties and conditional expression operators.

You can create as many conditions as required for a feature. For information on creating the conditions, see Setting Feature Conditions.

The Feature Condition Builder dialog box contains the following settings.

Table 12-27 • Feature Condition Builder Dialog Box Settings

Setting	Description
Conditions	This box displays a grid with the following columns: • **Level**—This column is where you assign the install level that should be used for the feature if the condition evaluates as true. • **Condition**—This column is where you define the condition. You can type a condition directly in this column, or you can use the Properties list, the Operators list, and the Add buttons to build a condition statement. To add a new condition to this box, click the New Condition button, and then enter the appropriate information in the Level and Condition columns. To delete a condition in this box, select the condition, and then click the Delete Conditions button.  ***Important*** • *When you click the OK button on this dialog box, InstallShield performs basic condition validation; however, you should still double-check that your condition statements evaluate to the expected outcome. For more information and example conditions, see Building Conditional Statements.*
Properties	This list contains common Windows Installer properties, as well as any properties that have been added in the Property Manager view. Select the property that you want to evaluate, and then click the **Add** button next to this list. If the list does not contain the property that you want to evaluate, you can type the appropriate property name in the Condition column.
Operators	This list contains all of the valid operators that are recognized by the Windows Installer. If your conditional statement should contain an operator, select the operator that you want to use and then click the **Add** button. InstallShield adds the operator to the conditional statement.

File Details Dialog Box

Access this dialog by clicking the Details button in the System Search Wizard. This button becomes active after you define a search method and specify the file you are locating. In this dialog, you have the ability to enhance a search by specifying the following details:

Table 12-28 • File Details Dialog Box Settings

Setting	Description
Minimum Version	The search is successful if the file exists on the target system and the version is equal to or higher than the version entered.
Maximum Version	The search is successful if the file exists on the target system and the version is equal to or lower than what is entered.
Minimum Date	Select the check box to search by a minimum date. The search is successful if the file exists on the target system and the date is equal to or later than the date entered.
Maximum Date	Select the check box to search by a maximum date. The search is successful if the file exists on the target system and the date is earlier or equal to the date entered.
Minimum Size	The search is successful if the file exists on the target system and is equal to or larger than the size indicated (in bytes).
Maximum Size	The search is successful if the file exists on the target system and is smaller than or equal to the size indicated (in bytes).
Languages	Click the Browse (...) button to display the Languages dialog. You can select multiple languages as a condition of your search. The search is successful if at least one of the languages listed is a match.



Note • *The information you provide in the edit fields is optional. You can leave any field blank.*

File Properties Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

- *Transform*

The Properties dialog box lets you override various properties for a file when it is installed on the target system.

Task ***To open the File Properties dialog box:***

1. In the View List under **Application Data**, click **Files and Folders**.
2. Right-click a file and click **Properties**.

Important • *File properties cannot be set for dynamically linked files.*

Table 12-29 • Properties Dialog Box Settings

Setting	Description
Location	This read-only setting displays the directory in which this file will be installed on a target system, based on the destination of the component that contains the file. To modify the destination, change the destination of the component that contains the file.
Font title	If you are installing a font, you can specify the font title in this box using the format **FontTitle (FontType)**—for example, **Roman (All res)**. InstallShield provides the name of the font if it is registered on your system. You should not specify a font title for .ttf or .ttc fonts because Windows Installer reads the embedded font name and registers it for you. Thus, for .ttf or .ttc files, InstallShield sets the value of this setting to **[Title read from file]**. For more information, see ICE07 in the Windows Installer Help Library.
Self Register	To self-register the file during installation, select this check box. The Self Register setting is valid for .dll, .ocx, .exe, .tlb, and .olb files. If the file is part of a 64-bit component, the installation performs 64-bit self-registration on the target system. For more information, see Targeting 64-Bit Operating Systems. ***Note*** • *It is recommended that you extract COM information from your self-registering files, instead of using self-registration. For more information, see Registering COM Servers.*

Table 12-29 • Properties Dialog Box Settings (cont.)

Setting	Description
Override system attributes	To install this file using the same system properties that are currently set for this file on the development system, clear this check box. To override one or more of the file's properties, select the **Override system attributes** check box and then select one or more of the following check boxes. • **Read-only**—If you want the file to be read-only when Windows Installer installs it, select this check box. • **Hidden**—If you want the file to be hidden when Windows Installer installs it, select this check box. • **Use File Hash**—This option is applicable for unversioned files only. Windows Installer can use file hashing to detect and avoid unnecessary copying of unversioned files. If you want Windows Installer to compare the hash of the file in your installation with the hash of the corresponding file on the target system when deciding whether to upgrade an existing file, select this check box. • **System**—If you want Windows Installer to install the file as a system file, select this check box. • **Vital**—To indicate that this file is vital to the operation of its component, select this check box. A vital file's component is not installed if the file cannot be installed for any reason. If a vital file cannot be installed, the end user sees an error message with Retry and Cancel buttons, instead of the usual Abort, Retry, and Ignore buttons (which enable the end user to complete the installation successfully without installing that file).  ***Note*** • *If No is selected for the Generate File Hash Values setting on the Build tab for a release in the Releases view, no file hash entries are created for any files, regardless of whether an individual file's Use File Hash check box is selected.*
Override system size	The Size setting shows the file's size when the **Override system size** setting is cleared. If you want Windows Installer to ignore the actual size of the selected file when it is calculating the required disk space requirements for your installation, and instead consider the size to be a value that you specify, select this check box and enter the appropriate value (in bytes) in the Size setting.

Table 12-29 • Properties Dialog Box Settings (cont.)

Setting	Description
Always Overwrite	If you want Windows Installer to ignore the actual version number of the selected file, and instead always try to overwrite the file if it is already present on the target system, select this check box. At run time, if this file has the same name and target location as one on the target system, Windows Installer considers the version of the file in your installation to be 65535.0.0.0 when it is determining whether to update the target system's file with the version in your current installation, or to leave the file as is. For more details about how Windows Installer determines whether to overwrite an existing file, see Overwriting Files and Components on the Target System. The maximum version number for a file is 65535.65535.65535.65535.
Override system version	If the file is a versioned file, the Version setting shows the file's version when the **Override system version** setting is cleared. If you want Windows Installer to ignore the actual version number of the selected file, and instead consider the version to be a number that you specify, select this check box and enter the appropriate version number in the Version setting. At run time, if this file has the same name and target location as one on the target system, Windows Installer uses the version number that you specify when it is determining whether to update the target system's file with the version in your current installation, or to leave the file as is. For example, if the file in your project is version 2.0.0.0, and you enter an override version of 3.0.0.0, Windows Installer may replace the file on the target system if it is version 3.0.0.1 or later, but not if it is earlier than 3.0.0.0. For more details about how Windows Installer determines whether to overwrite an existing file, see Overwriting Files and Components on the Target System. The maximum version number for a file is 65535.65535.65535.65535.
Override system language	If you want Windows Installer to ignore the language of the selected file and instead consider the language to be one that you specify, select this check box and type the decimal value for the language identifier code in the Language box. At run time, if this file has the same name and target location as one on the target system, Windows Installer considers the language of the file in your installation to be the one that you specified when it is determining whether to update the file with the version in your current installation, or to leave the file as is. For more details about how Windows Installer determines whether to overwrite an existing file, see Overwriting Files and Components on the Target System. Font files should not be authored with a language ID because fonts do not have an embedded language ID resource. Leave this entry blank for font files.
Permissions	To set permissions for the file, click this button.

File Properties Dialog Box (InstallScript Installation Projects)



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

This dialog box displays the properties that are set for a file when it is installed on the target system.



Task ***To open the File Properties dialog box:***

Right-click a file and click **Properties**. This option is disabled for files in components whose Link Type property is set to **Dynamic**.

In InstallScript projects, the File Properties dialog box displays information about the selected file. The information in the dialog box is taken directly from the file system and the dialog box is read only except for the following controls:

Register Font File

If this box is checked, the file is marked internally to be registered on the target machine if you have enabled global font registration. If this box is unchecked, the file is not registered on the target machine.

Font title

For an .fon file, this text specifies the font title that the installation registers for the font. By default, the InstallShield user interface reads the font title from the registry of the source system and enters that font title in this edit box. For a TrueType file (.ttf or .ttc file), by default no text is displayed in this edit box and the installation reads the font title from the file at run time. For all three types of files, if you enter non-default text in this edit box, the installation registers that text as the font title.

File Removal Properties Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The file removal Properties dialog box indicates information about a selected file or folder that you want to be removed from target systems. This file and folder removal capability is useful for scenarios such as removing application-created files that your installation does not otherwise track.

You can access the file removal Properties dialog box from within the Files and Folders view, or within the Setup Design view (for installation projects) or the Components view.

Task ***To access the file removal Properties dialog box from within the Files and Folders view:***

1. In the View List under **Application Data**, click **Files and Folders**.
2. In the **Destination computer's folders** pane, click the folder that contains the file removal item that you want to configure.
3. In the **Designation computer's files** pane, right-click the file removal item that you want to configure, and then click **Properties**.

   InstallShield uses an icon that has a red X through it for identifying files and folders that are configured to be removed.

Task ***To access the file removal Properties dialog box from within the Setup Design view (for installation projects) or the Components view:***

1. In the View List under **Organization**, click **Setup Design** (in Basic MSI, InstallScript MSI, or MSI Database projects only) or **Components**.
2. In the explorer, expand the component that contains the file removal item that you want to configure, and then click its **Files** node. In **Files** pane on the right, InstallShield displays the files and removal items that are associated with the selected component.
3. In the **Files** pane, right-click the file removal item that you want to configure, and then click **Properties**.

   InstallShield uses an icon that has a red X through it for identifying files and folders that are configured to be removed.

To learn how to add file and folder removal items to your project, see Removing Files and Folders from Target Systems.

Table 12-30 • Settings in the Properties Dialog Box

Setting	Description
Location	This setting indicates the location of the selected item that is configured to be removed.
Remove the folder if it is empty	If the selected removal item is a folder that you want to be removed from target systems when it is empty, select this option.
Remove files from the folder	If the selected removal item is a file in the folder that is identified in the Location setting and that you want to be removed from target systems, select this option.

Table 12-30 • Settings in the Properties Dialog Box (cont.)

Setting	Description
File Name	This setting is available if you select the **Remove files from the folder** option. Enter the name of the file that you want to be removed. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Removal Scheduling	Specify when you want the selected file or folder to be removed from target systems. Removals occur when the file or folder's associated component is processed in one of the following ways: • **Component installation**—The selected item is removed if its component is being installed and if—for folder removal items—the folder is empty. • **Component uninstallation**—The selected item is removed if its component is being uninstalled and if—for folder removal items—the folder is empty. • **Component installation and uninstallation**—The selected item is removed if its component is being installed or uninstalled and if—for folder removal items—the folder is empty.

Find and Replace Dialog Box

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The Find and Replace dialog box lets you search for strings in your SQL script file and replace them with different strings at run time. The Find and Replace dialog box opens when you click the Add button on the Text Replacement tab for a SQL script that is selected in the SQL Scripts view.

In Basic MSI, DIM, and InstallScript MSI projects, you can replace a string with a Windows Installer property; at run time, Windows Installer writes the value of the property in your SQL script file.

In InstallScript projects, you can use text substitution to replace a string variable with the appropriate value.

Caution • *You should only replace text that is unique and will not cause script syntax errors.*

The following settings are available on the Find and Replace dialog box.

Table 12-31 • Find and Replace Dialog Box Settings

Setting	Description
Find What	Enter the string that you want to replace in your SQL script at run time. ***Project*** *• In Basic MSI, DIM, and InstallScript MSI projects—You can use Windows Installer public properties to specify the replacement strings. To learn more, see Using Windows Installer Properties to Dynamically Replace Strings in SQL Scripts.* *In InstallScript projects—You can use InstallScript text substitution to specify the replacement strings. To learn more, see Using InstallScript Text Substitution to Dynamically Replace Strings in SQL Scripts.* *Specifying Windows Installer properties or InstallScript text substitution enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's SQL script is modified.*
Replace With	Enter the new string that should replace the existing string that is found in your SQL script at run time. ***Project*** *• In Basic MSI, DIM, and InstallScript MSI projects—You can use Windows Installer public properties to specify the replacement strings. To learn more, see Using Windows Installer Properties to Dynamically Replace Strings in SQL Scripts.* *In InstallScript projects—You can use InstallScript text substitution to specify the replacement strings. To learn more, see Using InstallScript Text Substitution to Dynamically Replace Strings in SQL Scripts.* *Specifying Windows Installer properties or InstallScript text substitution enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's SQL script is modified.*
Match case	This check box lets you indicate whether you want to restrict your search to strings with the same capitalization that you use in the Find What setting. For example, if you select this check box and enter **install** for the Find What setting, the installation searches only for all-lowercase instances of that string. If you clear this check box, the installation searches for *install*, as well as *INSTALL*, *Install*, and other mixed-case instances.

Table 12-31 • Find and Replace Dialog Box Settings (cont.)

Setting	Description

Match whole word only	This check box lets you indicate whether you want to find only whole-word instances of the value that you have entered for the Find What setting.
	
	For example, if you select this check box and enter **install** for the Find What setting, the installation searches only for instances of *install*; it does not search for other forms of the word, such as *installs*, *installation*, or *uninstall*.
	
	For another example, you may have something like the following line in your SQL script:
	
	`PASSWORD = N'%PASSWORD%'`
	
	If you select this check box, enter **%PASSWORD%** in the Find What setting, and enter **1234** in the Replace With setting, the installation replaces the string. The resulting SQL script shows the following:
	
	`PASSWORD = N'1234'`

Preserve case	This check box lets you indicate whether you want the replacement string to use the same capitalization that is used in the Find What string. For example, if you select this check box and enter **databasedir** for the Find What setting and **mydatabasedir** for the Replace With setting, the installation replaces *Databasedir* with *Mydatabasedir*. It also replaces *databasedir* with *mydatabasedir*.
Replace once only	This check box lets you indicate whether you want the installation to replace only the first occurrence of the search string. If you clear this check box, the installation replaces all instances of the search string.

Find String ID in Project Dialog Box

The Find String ID dialog box lets you search your entire project for occurrences of a particular string entry. Select from the list the string entry you would like to find and click OK. The results of your search are displayed in the Output window at the bottom of the screen.

Folder Properties Dialog Box

A folder's Properties dialog box determines the properties that are set for a folder when it is installed on the target system. To access this dialog box, right-click a folder in the Files and Folders view, and then click Properties.

The Properties dialog box is organized into the following tabs:

- General
- Sharing

General Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The General tab on a folder's Properties dialog box lets you configure basic properties about a folder.

Table 12-32 • General Tab Settings

Setting	Description
Directory Identifier	This setting identifies the current directory identifier for the selected folder.
Source Location	This setting identifies from where files will be copied to the disk image folders when you build an uncompressed release.
Target	This setting identifies the location to which the folder will be installed on target systems.

The Permissions button lets you configure permissions for the folder. To learn more, see Permissions Dialog Boxes for Files and Directories.

Sharing Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

The General tab on a folder's Properties dialog box lets you configure basic properties about a folder.

Table 12-33 • General Tab Settings

Setting	Description
Share this folder on the network	To share this folder on the network, select this check box.
Share name	If you are enabling sharing for this folder, specify the name that users need to use to connect to the shared folder.
Limit the number of simultaneous users to	If you are enabling sharing for this folder, specify the maximum number of users who can connect to the shared folder at one time. ***Note*** • *Windows imposes limits for the maximum number of simultaneous users. These limits vary, depending on the version and edition of Windows. An installation cannot override this Windows-imposed limit. Thus, if you specify a larger number than a target system can support, the installation sets the limit at the maximum that is allowed by that target system's operating system.* *To determine the limit for a specific version and edition of Windows, use the following command-line statement on that version and edition of Windows:* `Net Config Server` *The Maximum Logged On Users value that the Command Prompt window displays indicates the maximum that the operating system supports.* *You can also find the maximum number in the "Device Connections" section of a Windows end-user license agreement.*
Allow network users to change my files	If you are enabling sharing for this folder and you want to allow network users to change the files in this folder, select this check box.
Component	Select the component in this folder that should include this network share.

Function Signature Dialog Box

The Function Signature dialog box is where you specify the parameters for the entry-point function in your standard .dll file.

Task ***To access the Function Signature dialog box:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in DIM, Merge Module, and MSM Database projects).
2. In the **Custom Actions** explorer, select the custom action whose parameters you would like to specify.

3. In the **Function Signature** property, click the ellipsis button (**...**).

Dialog Box Options

Name

Specify the name of the function that you want to call. The .dll file can have a single entry point for each custom action.

Arguments

Build the list of arguments—in order—that you want to pass to the function.

First, click the last row of the Arguments list to add a new argument. Next, complete the Type, Source, and Value columns for each argument.

Type

In this list, select the data type of the parameter. Note that there are two special cases for argument types, and these cases are identified below.



Task ***To initialize a pointer to null:***

1. In the Type list, select **POINTER**.
2. In the **Source** list, select **Constant**.
3. In the **Value** list, select **NULL**. Do not enter 0 (zero) instead of **NULL**, because it is interpreted as a pointer pointing to the number 0.



Task ***To set a handle to the .msi database or the Windows Installer window:***

1. In the **Type** list, select **HANDLE**.
2. In the **Source** list, select **Constant**.
3. In the **Value** list, select **MsiHandle** or **MsiWindowHandle**, depending whether you want a handle to the .msi database or the Windows Installer window.

Source

Indicate how you want to pass the data to the function. The following options are available in this list.

Table 12-34 • Function Signature Dialog Box Options

Option	Description
Constant	The value that you want to pass is stored as a literal in your installation project. After you select **Constant** for the argument's source, enter its definition in the Value column. This constant does not require a name or identifier. Do not enclose string literals in quotation marks.

Table 12-34 • Function Signature Dialog Box Options (cont.)

Option	Description
in Property	Select **in Property** for the source to specify the name of a Windows Installer property whose value will be passed to the function. This type of argument is suitable for a ByVal parameter.
inout Property	This source type is similar to a ByRef parameter. Select **inout Property** for the source to specify the name of a Windows Installer property whose value will be passed to the function and can be modified by the function.
out Property	An **out property** serves as a placeholder for a value that can be set by the function. No value is passed to the function, but the function requires the name of a property whose value it can modify.

When you select a property for the argument's source, you must specify the name of the property in the Value column.

Value

The value can be one of two types:

- A number or string literal that you enter when the argument's source is a constant
- A Windows Installer property when the argument's source is a property



Note • *When you set the Type list to HANDLE and the Source list to Constant, the Value column contains two options: MsiHandle and MsiWindowHandle. These constants can be used to get a handle to either the .msi database or the Windows Installer window.*

When the source is a property, the Value list provides the name of every property in the Property Manager. You can select an existing property or enter a new name, in which case the new property is added to the Property Manager.

Remember that a property is merely a container for a value. If your parameter stores its value in an in property or inout property, ensure that you specify a value for the property in the Property Manager.

(Context Menu)

The context menu provides you with options for working with your arguments. To access the context menu, right-click an argument in the Arguments grid. The context menu options are described below.

Table 12-35 • Context Menu Options

Option	Description
Add	Click Add to add an argument to the Arguments grid.
Remove	Click Remove to delete the argument.
Move Up	The arguments must be in the precise order in which the function expects to receive them. Click Move Up to place the argument higher in the list.

Table 12-35 • Context Menu Options

Option	Description
Move Down	Click Move Down to place the argument lower in the list.

Return Type

Select the data type of the function's return value. If you do not need to check the return value, you can accept void.

Return Property

Select the name of a property whose value will be set to the function's return value. If you are going to check the return value elsewhere in your installation, you may want to initialize the return property's value.

Note • *The property cannot be set to the function's return value unless the action is configured for immediate execution. Thus, if the action is scheduled for deferred, rollback, or commit execution, the Return Property setting is ignored at run time.*

General Options - Advanced Dialog Box

The General Options - Advanced dialog box opens when you click the Advanced button on the General Options panel of the Release Wizard. This dialog box allows you to specify a custom location for the built files, (in installation projects only) the space that is reserved in the disk images folders, whether the installation performs MD5 checking, and (in object projects only) how InstallShield handles subobjects contained in the object project during the build of the object.

Panel Options

Build the media to the custom folder location

If unchecked, the release's Disk Images, Log Files, and Report Files folders are created in the default location:

<project folder>\Media\<release name>

Check this box to specify another location by typing in the combo box, selecting and completing one of the list items, which are defined relative to path variables, or clicking the browse button and selecting a folder.

Reserved Space

Project • *This option is available in installation projects. If you are building an object project, this setting is not displayed.*

Disabled if the current release's media type produces a single disk image—which is true for Network Image. Lets you reserve (leave empty) specified amounts of space in particular disk image folders. Select a disk image folder number in the Disk list box (for example, a value of 1 specifies disk image folder Disk1) and enter the space (in kilobytes) to be reserved in that folder in the Reserved Space edit box.

Verify MD5 signature



Project • *This option is available in installation projects. If you are building an object project, this setting is not displayed.*

If this check box is checked, the setup compares each installed file's MD5 hash value to the value stored in the .cab file. This setting is stored in the CheckMD5 key of the [Startup] section in the Setup.ini file.



Tip • *MD5 checking can detect corrupted files, which is useful during Internet setups; not doing MD5 checking can make file transfer proceed faster.*

Object Inclusion - Embedded



Project • *This option is available in object projects.*

If the object project includes any subobjects, the release includes those subobjects themselves rather than references to the subobjects. A release built with this option can be used by InstallShield users who do not have the subobjects registered on their systems.

Object Inclusion - Reference



Project • *This option is available in object projects.*

If the object project includes any subobjects, the release includes references to those subobjects rather than the subobjects themselves. A release built with this option cannot be used by InstallShield users who do not have the subobjects registered on their systems.

General Options - Other Disk Files Dialog Box

The **General Options - Other Disk Files** dialog box opens when you click the Other Disk Files button on the General Options panel of the Release Wizard. This dialog box allows you to specify a location in the disk image folders for each file that is in the Other folder under Advanced Files in the Support Files/Billboards view.

Panel Options

File name

Select the file for which you want to specify a disk image folder number.

Disk

Select the desired disk image folder for the selected file (for example, selecting 3 places the selected file in disk image folder Disk3).

History Dialog Box

The History dialog is displayed when you select **Show History from the Project/Source Control** menu. The dialog contains a drop-down list box that contains all of the Windows Installer tables in your setup project.



Task ***To display view the table's history in your source control program:***

Select a table from the list and click **OK**.

Import Dialog Dialog Box

The Import Dialog dialog box lets you select a dialog that you want to import into your installation project. The file for the dialog (.isd file) can be stored in a repository, or it can be stored in some other location.



Task ***To access the Import Dialog dialog box:***

1. Open the **Dialogs** view.
2. In the Dialogs explorer, right-click **All Dialogs** and then click **Import Dialog.** The **Import Dialog** dialog box opens.

Dialog Box Options

Repository Items

This box lists all of the dialogs that are available in the repository. Select the dialog that you want to add to your project.

If the file for the dialog box (.isd file) that you want to import is not stored in the repository, click the Browse button to select it.

Import InstallScript Files Dialog Box

The Import InstallScript Files dialog box lets you select one or more InstallScript (.rul) or InstallScript header (.h) files that you want to import into your installation project. The files can be stored in a repository, or they can be stored in some other location.



Task ***To access the Import InstallScript Files dialog box:***

1. Open the **InstallScript** view.
2. In the InstallScript explorer, right-click **Files** and then click **Import Script File**. The **Import InstallScript Files** dialog box opens.

Dialog Box Options

Repository Items

This box lists all of the InstallScript files that are available in the repository. Select one or more InstallScript files that you want to add to your project.

If the InstallScript file that you want to import is not stored in the repository, click the Browse button to select it.

Import SQL Script Files Dialog Box

The Import SQL Script Files dialog box lets you select one or more SQL script files that you want to import into your installation project. The SQL script (.sql) file can be stored in a repository, or it can be stored in some other location.

Task ***To access the Import SQL Script Files dialog box:***

1. Open the **SQL Scripts** view.
2. In the SQL Scripts explorer, right-click Files and then click **Import Script File**. The **Import SQL Script Files** dialog box opens.

Dialog Box Options

Repository Items

This box lists all of the SQL script files that are available in the repository. Select one or more SQL script files that you want to add to your project.

If the SQL script file that you want to import is not stored in the repository, click the Browse button to select it.

Insert Action Dialog Box

Project • *The Insert Action dialog box is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Insert Action dialog box enables you to insert a standard action, custom action, or dialog into one of your project's sequences.

Task ***To launch the Insert Action dialog box:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences**.
2. Right-click a sequence, dialog, custom action, or standard action, and then click **Insert**.



Tip *• You can also launch this dialog box from the DIM References view in a Basic MSI project: In this view, click a DIM reference, and then click the Sequences tab. Right-click a sequence, dialog, custom action, or standard action, and then click Insert.*

Dialog Options

Table 12-36 • Insert Action Dialog Box Options

Option	Description
Action Type	Select the type of action to insert: • **Custom Actions**—You can create custom actions with the Custom Action Wizard. • **Dialogs**—You can insert dialogs into the User Interface sequence. To create custom dialogs, use the Dialogs view. • **DIM Custom Actions**—These custom actions are stored within one or more DIM references that you have added to your installation project. The GUID for that DIM is appended to the name of the custom action. For example, if you have a LaunchUtility action called LaunchUtility stored within a DIM, it appears as LaunchNotepad.xxxxxxxx_xxxx_xxxx_xxxx_xxxxxxxx, where x is the GUID for that DIM. • **DIM Dialogs**—These dialogs are stored within one or more DIM references that you have added to your installation project. The GUID for that DIM is appended to the name of the dialog. For example, if you have a dialog called CustomerInfo stored within a DIM, it appears as CustomerInfo.xxxxxxxx_xxxx_xxxx_xxxx_xxxxxxxx, where x is the GUID for that DIM. • **Merge Module Custom Actions**—These custom actions are stored within one or more merge modules that you have associated with your installation project. The GUID for that merge module is appended to the name of the custom action. For example, if you have a custom action called LaunchNotepad stored within a merge module, it appears as LaunchNotepad.xxxxxxxx_xxxx_xxxx_xxxx_xxxxxxxx, where x is the GUID for that module. • **Merge Module Dialogs**—These dialogs are stored within one or more merge modules that you have associated with your installation project. The GUID for that merge module is appended to the name of the dialog. For example, if you have a dialog called CustomerInfo stored within a merge module, it appears as CustomerInfo.xxxxxxxx_xxxx_xxxx_xxxx_xxxxxxxx, where x is the GUID for that module. • **Standard Actions**—These actions are built into Windows Installer.  ***Note*** • *When you delete a merge module from your project that contains custom actions or dialogs inserted within one of your sequences, InstallShield removes those actions and dialogs automatically from the sequence. The same is true for DIM references.*
Select an Action	After you have selected the action type, you need to select the specific action. To do this, highlight the action you want to insert.

Table 12-36 • Insert Action Dialog Box Options (cont.)

Option	Description
Condition	You can set a condition upon this action. If the condition does not evaluate to true, the action does not execute. For more information on conditions, see Conditional Statement Syntax.
Comments	Enter comments in this field. These comments are for internal use only and are not displayed to the end user.

InstallShield Prerequisite Properties Dialog Box

The InstallShield Prerequisites Properties dialog box opens when you right-click a selected InstallShield prerequisite in the Redistributables view or in the Prerequisites view and then click Properties. This dialog box enables you to specify a location for the selected InstallShield prerequisite. To learn more, see Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level.



Project • *In Basic MSI and InstallScript MSI projects, the InstallShield Prerequisite Properties dialog box also enables you to set release flags for InstallShield prerequisites that you want to exclude from certain builds. For example, if you*

have an InstallShield prerequisite that should be included only in a special edition of your product that contains a special add-on that requires the prerequisite, you can flag that prerequisite and include it only when it is needed.

Table 12-37 • InstallShield Prerequisite Dialog Box Settings

Setting	Description
Build Location	When you package an installation that includes InstallShield prerequisites, you can use any one several methods for supplying the prerequisite files to end users: • **Download From The Web**—Download the InstallShield prerequisite files (if necessary) from the URL that is specified for this InstallShield prerequisite . This option is recommended if your installation will be downloaded from the Internet and you want to minimize the package size and download time. An InstallShield prerequisite will not be downloaded if the correct version is already present on the target machine. This option is available for Basic MSI, InstallScript, and InstallScript MSI projects. • **Extract From Setup.exe**—Compress the InstallShield prerequisite files into **Setup.exe**, to be extracted at run time, if necessary. Select this option if the entire installation must be self-contained in **Setup.exe**. Note that the Download From The Web option results in smaller installations and shorter download time; however, the Extract From Setup.exe option provides for a completely self-contained installation. This option is available for Basic MSI and InstallScript MSI projects. • **Copy From Source Media**—Store the InstallShield prerequisite files on the source media. Use this option if the installation will be run uncompressed from fixed media—CD, DVD, or local network. This option is available for Basic MSI and InstallScript MSI projects. • **Include with Media**—Store the InstallShield prerequisite files on the source media. This option is available for InstallScript projects. Note that the option that you select in the Build Location setting may be overridden in the Releases view. For more information, see Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level. ***Note*** • *Note that if an InstallShield prerequisite is added to a project as a dependency of another prerequisite, the location for the prerequisite dependency follows the location setting of the prerequisite that requires it.*
Release Flags	***Project*** • *This setting is available in Basic MSI and InstallScript MSI projects.* Type a string. The string can be any combination of letters or numbers. To have more than one flag on an InstallShield prerequisite, use a comma to separate the flags

Languages Dialog Box



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The Languages dialog box opens when you click the ellipsis button (...) for the Languages setting in the Components view. This dialog box enables you to specify the languages for which the component is intended, for use in build-time filtering. By default, components are language independent, meaning that none of the component's data (such as files and registry entries) are specific to a particular language.

To designate that the component is intended for only certain languages, select the check boxes for the appropriate languages.



Project *• In InstallScript and InstallScript Object projects, if a language is not selected in the General Information view of a project, it is not listed as one of the available languages for the project's components.*

Link Type Dialog Box



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The Link Type dialog box opens when you click the browse button in the Link Type setting in the Components and Setup Design views. It also opens when you right-click a component in the **Destination computer's folders** pane in the Files and Folders view and then click File Linking.

The Link Type dialog box lets you specify whether the file links of the selected component are static or dynamic and, if the latter, how the links are determined at the time of the media build.

Dialog Box Settings

Static Links

The component contains file links to files specified by explicit fully qualified file names.

Dynamic Links

The component contains file links to files whose names can be specified by wild cards (for example, *.* or *.exe) and whose folder can be specified by an explicit path or by a build variable.



***Note** • The following settings are enabled only if the Dynamic Links option is selected.*

Specify the folder which contains the files to include

This setting specifies the folder that contains the linked files. Type an explicit path, select or type a build variable, or select and optionally append to a predefined path variable.



***Caution** • Entering a relative path (for example, .\MyFolder) will give unpredictable results.*

Browse

Clicking this button opens the Browse for Folder dialog box, in which you can specify an explicit path.

Include subfolders

If you select this check box, InstallShield dynamically links the files that are included in any subfolders of the folder that you identified for the **Specify the folder which contains the files to include** setting.



***Tip** • The InstallScript engine does not support the concept of subcomponents. When you include subfolders in a dynamic file link, InstallShield creates separate components for each subfolder.*

Wildcard(s)

The file list is constructed by first including all files that match the specification you enter in the Inclusion combo box, and then excluding from those files all the ones that match the specification you enter in the Exclusion combo box.

Inclusion

Specify the names of the files to which the component contains file links. By typing an entry or selecting a path variable, you can enter the name of a single file or use wild cards in this filename specification, for example *.* or *.exe. To specify multiple wild cards, separate them with semicolons and no spaces, for example,

.txt;.doc

Exclusion

Specify the names of the files to which the component does not contain file links. By typing an entry or selecting a path variable, you can enter the name of a single file or use wild cards in this filename specification, for example *.* or *.exe. To specify multiple wild cards, separate them with semicolons and no spaces, for example,

.exe;.dll

In the above example, no .exe or DLL files are included in the component.

Logging Options for Windows Installer 4.0 and Later Dialog Box

InstallShield displays the Logging Options for Windows Installer 4.0 and Later dialog box when you click the ellipsis button (...) for the Create MSI Logs setting in the General Information view. This dialog box enables you to specify on a project-wide basis—without having to use the command line or configure log parameters through the registry—whether Windows Installer should log your installation. You can also use this dialog box to customize the types of messages that are logged.

Table 12-38 • Options for the Logging Options for Windows Installer 4.0 and Later Dialog Box

Option	Description
No	Installations are not logged. This is the default value.
Yes (MsiLogging set to default value of voicewarmupx)	InstallShield populates the **MsiLogging** property with the default value of **voicewarmupx**. If the installation is run on a target system that has Windows Installer 4.0 or later, as well as Windows Vista or later or Windows Server 2008 or later, the following occurs: • The installer creates a log file according to the default logging mode of voicewarmupx. • The installer populates the **MsiLogFileLocation** property with the log file's path. • A **Show the Windows Installer log** check box is added to the SetupCompleteSuccess, SetupCompleteError, and SetupInterrupted dialogs. If the end user selects that check box and then clicks Finish, the log file is opened in a text file viewer or editor. This setting applies to installations that are run with Windows Installer 4.0 or later on Windows Vista and later systems or Windows Server 2008 and later systems. The **Show the Windows Installer log** check box is not visible in run-time dialogs that are displayed on earlier systems that are running earlier versions of Windows Installer.

Table 12-38 • Options for the Logging Options for Windows Installer 4.0 and Later Dialog Box (cont.)

Option	Description
Custom MsiLogging value	InstallShield populates the **MsiLogging** property with the value that you specify in the box. If the installation is run on a target system that has Windows Installer 4.0 or later, as well as Windows Vista or later or Windows Server 2008 or later, the following occurs: • The installer creates a log file according to the custom value that you specified in the box. • The installer populates the **MsiLogFileLocation** property with the log file's path. • A **Show the Windows Installer log** check box is added to the SetupCompleteSuccess, SetupCompleteError, and SetupInterrupted dialogs. If the end user selects that check box and then clicks Finish, the log file is opened in a text file viewer or editor. This setting applies to installations that are run with Windows Installer 4.0 or later on Windows Vista and later systems and Windows Server 2008 or later systems. The **Show the Windows Installer log** check box is not visible in run-time dialogs that are displayed on earlier systems that are running earlier versions of Windows Installer.



Important • *The **MsiLogFileLocation** property is read-only; it cannot be used to set or change the log file location.*

Lowercase Component Directory Warning

This dialog appears when you declare a Directory Identifier in lowercase or in mixed-case letters when setting a component's destination.

Declaring a Directory Identifier in uppercase letters makes that value a public property that can be set from the user interface. Using mixed-case or lowercase letters for the Directory Identifier defines the directory entry as a private property. Private properties cannot be changed from the user interface.



Note • *To suppress this warning, select the **Do not show this dialog again** option.*

Media File Properties Dialog Box

Opens when you click the Import or Modify button in the Release Wizard's Update panel. Lets you specify a release that is not in the current project and the method for determining its version information.

Media Header File

Type, or select and complete, the fully qualified name of the desired release's header file (such files have extension .hdr and are typically named Data1.hdr), or click the browse button to browse for the header file.

Read the version information ...

Specifies that the version information is automatically read from the release's header file.

Specify the version information below

Specifies that the version information is what you type or select in the combo box.

Merge Module Configurable Values Dialog Box

A configurable redistributable is a merge module or an object that has at least one row in the **ModuleConfiguration** table that is referenced by at least one row in the **ModuleSubstitution** table. This allows you to change a value in the redistributable.

Displaying the Merge Modules Configurable Values Dialog Box

The Merge Module Configurable Values dialog box opens when you select a configurable merge module or object whose check box is selected in the Redistributables view. If the redistributable selected is an object, the Merge Module Configurable Values dialog box is displayed when the object wizard finishes.

You can also right-click a configurable merge module or object and select Configure Merge Module.

Dialog Box Settings

The Merge Module Configurable Values dialog box contains a grid in which you can modify the configurable values. The left column contains the name of the configurable value. The right column contains the value options. You can select a new value from the drop-down menu. Refer to the redistributable vendor's documentation for information about the values.

After you specify the values that you want, click OK to save the new values. When you build your project, these values are used to build the .msi package.

Restore Defaults

Click this button to restore the redistributable's default settings. All of the configurable values in the redistributable are returned to their defaults.

Merge Module Properties Dialog Box

The Merge Module Properties dialog opens when you right-click a selected merge module or object in the Redistributables view and then click Properties. The following tabs are associated with this dialog box:

- Media
- Destination
- Configurable Values

Media Tab

File Location

Use the Media tab to specify the destination and format of your output.

On the source media (uncompressed)

Choose this option to copy the files uncompressed to the media.

In a new .CAB file

When you choose this option, InstallShield puts the source media in the same location as the .msi package. However, you can choose to stream the .cab into the .msi package.

Stream the new .CAB file into the Windows Installer Package

Choose this option to stream the .cab into the .msi package.



***Note** • For transform (*.mst) project types, the option to stream the .cab into the .msi package is disabled because the transform project cannot store the .cab fie internally. It can only add uncompressed files or files stored in an external cab.*

Destination Tab

Use the Destination tab on the Merge Module Properties dialog box to configure destination information.

Settings

GUID

Displays the merge module's unique GUID.

Author

Displays the merge module's author.

Version

Displays the merge module version.

Destination

Specifies the destination of the merge module's files. It is recommended you use the default setting "(Use merge module's default destination)".

Method Browser Dialog Box

The Method Browser dialog box is where you specify the class and method in your managed assembly that your custom action should invoke. The Custom Action Wizard displays this dialog box when you click the Browse button on the Managed Method Definition panel. In addition, this dialog box is also displayed if you click the Browse button on the Method Signature dialog box.



***Note** • Only public methods of public classes are available for selection. In addition, overloaded methods cannot be selected.*

*The browse functionality is available only if the .NET Framework is installed. In addition, it is available only if the location of the managed assembly is set to the **Binary** table or installed with the product—not if the assembly's path references a property or a directory in the **Directory** table.*

Also note that the browse functionality is available for only 32-bit assemblies and for Microsoft intermediate language (MSIL) files. It is not available for 64-bit assemblies.

Method Signature Dialog Box

InstallShield displays the Method Signature dialog box when you click the ellipsis button (...) in the Method Signature setting for a managed custom action in the Custom Actions and Sequences view (or the Custom Actions view). This dialog box lets you specify the public class method that you want your custom action to invoke.

The Method Signature dialog box has the following settings.

Table 12-39 • Settings on the Method Signature Panel

Setting	Description
Class	Enter the name of the public class—including the full namespace—with which the method is associated. Use the following format: **MyNamespace.MyClass** As an alternative, click the Browse button to display the Method Browser dialog box. This dialog box lets you select the public method from the list of public classes that are available in the managed assembly that you selected on the Action Parameters panel.  ***Note** • The browse functionality is available only if the .NET Framework is installed. In addition, it is available only if the location of the managed assembly is set to the **Binary** table or installed with the product—not if the assembly's path references a property or a directory in the **Directory** table.* *Also note that the browse functionality is available for only 32-bit assemblies and for Microsoft intermediate language (MSIL) files. It is not available for 64-bit assemblies.*

Table 12-39 • Settings on the Method Signature Panel (cont.)

Setting	Description
Method	Enter the public method that you want your installation to call. As an alternative, click the Browse button to display the Method Browser dialog box. This dialog box lets you select the public method from the list of public classes that are available in the managed assembly that you selected on the Action Parameters panel. ***Note*** • *The browse functionality is available only if the .NET Framework is installed. In addition, it is available only if the location of the managed assembly is set to the* ***Binary*** *table or installed with the product—not if the assembly's path references a property or a directory in the* ***Directory*** *table.* *Also note that the browse functionality is available for only 32-bit assemblies and for Microsoft intermediate language (MSIL) files. It is not available for 64-bit assemblies.*
Use custom method signature	To use the default method signature, clear this check box. For the default method signature, the installation calls the method with an MsiHandle parameter if the signature has one or more void parameters, or if it has a single MsiHandle parameter. To use a custom method signature, select this check box. Then specify the arguments and return property as needed. If you specified a class and a method by clicking the Browse button, InstallShield automatically adds any associated parameters. Specify the value for each parameter that you want to pass to the method. For detailed information about specifying the method's signature, see Specifying the Signature for a Managed Method in an Assembly Custom Action.

Table 12-39 • Settings on the Method Signature Panel (cont.)

Setting	Description
Arguments	If you are using a custom method signature, specify the list of arguments that you want to pass to the selected method. If you specified a class and a method by clicking the Browse button, InstallShield automatically adds any associated parameters to the grid in this area. To use a property as an argument, click the value field for the parameter, and then select the property from the list. You can also use a property that is not included in the list by typing the name of the property. The property must be defined at run time. The context menu provides you with options for working with arguments. To access the context menu, right-click an argument in the Arguments grid. The available context menu commands are: • **Add**—Add an argument to the grid. • **Remove**—Delete an argument from the grid. This grid is available only if the **Use custom method signature check box** is selected. For more information about specifying the arguments, see Specifying the Signature for a Managed Method in an Assembly Custom Action.
Return Property	Select or type the name of the property whose value will be set to the method's return value. This list is available only if the **Use custom method signature check box** is selected. For more information about specifying the return property, see Specifying the Signature for a Managed Method in an Assembly Custom Action.

MIME Types Dialog Box

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The MIME Types dialog box lets you add, edit, and delete mappings between file name extensions and the corresponding content types that are served as static files by the Web server to a browser or mail client.

The MIME Types dialog box is available from within the Internet Information Services view. To open this dialog box, click a Web site, application, or virtual directory in the explorer. Then click the ellipsis button (...) in the MIME Types setting.

Table 12-40 • MIME Types Dialog Box Settings

Setting	Description
Add	To add an entry for mapping a file name extension and the corresponding content type that is served as a static file by the Web server to a browser or mail client, click this button. This opens the Add MIME Types dialog box.
Edit	To edit an existing MIME type, select the MIME type and click this button.
Delete	To delete an existing MIME type, select the MIME type and click this button.

Modify Dynamic Links Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The Modify Dynamic Links dialog box shows all of the settings for your dynamic link.



Task ***To launch the dialog box:***

1. Open the **Components** view or the **Setup Design** view.
2. In the explorer, expand the component whose dynamic link you would like to modify and then click Files.
3. Right-click in the Files pane and then click **Dynamic File Linking**. The **Modify Dynamic Links** dialog box opens.

Table 12-41 • Settings on the File Linking Tab of the Component Properties Dialog Box

Setting	Description
Source	This column shows where the source files are stored. A path variable is typically used.
Include SubFolders	This column indicates whether InstallShield should dynamically link the files in each subfolder. For information on how InstallShield creates components for the dynamically linked files in subfolders, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.

Table 12-41 • Settings on the File Linking Tab of the Component Properties Dialog Box (cont.)

Setting	Description
Self-Register	This column indicates whether you want InstallShield to self-register every file in the dynamic link. If the file are part of a 64-bit component and the Self-Register column indicates Yes for a dynamic link, the installation performs 64-bit self-registration on the target system. For more information, see Targeting 64-Bit Operating Systems.
Create Components	This column indicates how you want InstallShield to create the components when it adds the dynamic files to your release at build time. The valid options are: • **Best Practice**—InstallShield adheres to best practices when creating components for the dynamically linked files. • **By Directory**—InstallShield creates one component for each folder and subfolder. To learn about these two methods, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.
Details	This column indicates the filter criteria for including files in and excluding files from the dynamic link.



Tip • *Note the following guidelines for working with the File Linking tab:*

- *To add a new dynamic link, click the New Link button. InstallShield displays the Dynamic File Link Settings dialog box.*
- *To modify the settings for an existing link, select the link and then click the Modify button. InstallShield displays the Dynamic File Link Settings dialog box.*
- *To remove a link and its associated dynamically linked files, select the link and then click the Delete button.*

Modify Property Dialog Box/Release Wizard—Platforms Panel



Project • *The following information applies to InstallScript projects.*

The Modify property dialog box and the Platforms panel in the Release Wizard let you specify whether a release is specific to one or more platforms. If the platform specified for a component does not match one of the platforms that is selected for the release, the component is not included in the release.

Task ***To access the Modify Property dialog box:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, click the release that you want to configure.
3. Click the **Build** tab.
4. Click the value of the **Platform(s)** setting, and then click the ellipsis button (**...**).

Task ***To access the Platforms Panel in the Release Wizard:***

1. In the View List under **Media**, click **Releases**.
2. In the **Releases** explorer, right-click the release that you want to configure, and then click **Release Wizard**.
3. Complete each of the panels in the wizard until you reach the **Platforms** panel.

The following table shows the settings that are displayed on the Modify Property dialog box and the Platforms panel in the Release Wizard.

Table 12-42 • Settings on the Modify Property Dialog Box and the Platforms Panel in the Release Wizard

Setting	Description
Use platforms specified by the Platform Filtering setting	If the selected release supports all of the platforms that are specified for the Platform Filtering setting in the General Information view, select this option. If you select this option, none of the components are excluded from the release at build time.
Specify the platforms to support below	If the selected release is specific to one or more operating systems, select this option and then select the appropriate operating systems. If the platform specified for a component does not match one of the platforms that is selected for this setting, InstallShield does not include the component in the release. ***Tip*** • *To select multiple consecutive operating systems, select the first file, press and hold SHIFT, and select the last operating systems. To select multiple nonconsecutive operating systems, select the first operating systems, press and hold CTRL, and select each additional operating systems.* ***Project*** • *For InstallScript projects, the list of operating systems that are displayed in this dialog box excludes any of the operating systems whose check box is cleared at the project level.*

Module Dependencies Dialog Box



Project *• This information applies to the following project types:*

- *Merge Module*
- *MSM Database*

The Module Dependencies dialog box opens when you click the ellipsis button (...) in the Module Dependencies setting of the General Information view. This dialog box lets you select the merge modules that are required for the merge module that you are creating.



Tip *• The dialog box lists the merge modules that are found in the directories that are specified on the Merge Modules tab of the Options dialog box. If you want to add a dependency that is not present on your machine, you can do so by clicking the* ***Add a new module dependency*** *button in the Module Dependencies setting of the General Information view.*

Module Exclusions Dialog Box



Project *• This information applies to the following project types:*

- *Merge Module*
- *MSM Database*

Some merge modules do not operate correctly in the presence of certain other merge modules. This may occur because an earlier version of a module might not be compatible with your new module. As a result, you need to ensure that the incompatible module is excluded from any installation that will contain your new module.

The Module Exclusions dialog box opens when you click the ellipsis button (...) in the Module Exclusions setting of the General Information view. This dialog box lets you select the merge modules that are compatible with the merge module that you are creating.

If you add a merge module that has exclusions to an installation project, and those excluded modules have already been added to your installation project, InstallShield silently removes any reference to the exclusions from your project.



Caution *• Any excluded modules that are added to the installation project after the excluding module has been added will not be removed; in addition, you will not receive any warning that they are incompatible.*



Tip *• The dialog box lists the merge modules that are found in the directories that are specified on the Merge Modules tab of the Options dialog box. If you want to add an exclusion that is not present on your machine, you can do so by clicking the* ***Add a new module dependency*** *button in the Module Exclusions setting of the General Information view.*

MSI Value Dialog Box

The MSI Value dialog box enables you to create or modify a primary key for an entry in the resulting Windows Installer package's **Registry** table. For more information, see Specifying a Primary Key for the Registry Table.

MST SIS Settings Dialog Box

This dialog allows you suppress errors and specify validation options for the transform and base MSI package. For more information, see Database.CreateTransformSummaryInfo in the Windows Installer Help Library.

Dialog Options

Suppress Errors Options

Select the check boxes next to the error conditions that you want to suppress.

Table 12-43 • Suppress Errors Options

Condition	Description
Add existing row	Adds a row that already exists.
Add existing table	Adds a table that already exists.
Delete missing row	Deletes a row that does not exist.
Delete missing table	Deletes a table that does not exist.
Modify missing row	Updates or modifies a row that does not exist.
Change codepage	The transform and .msi code pages do not match and neither code page is neutral.

Validation Options

In this section, you can indicate the information that you want the transform package to validate on the base MSI package before the transform is applied.

Table 12-44 • Validation Options

Validation Option	Description
Match languages	Validates the languages in both the base MSI and the transform. The transform is not applied if the languages do not match.
Match Upgrade Code	Validates the Upgrade Codes in both the base MSI and the transform. The transform is not applied if the Upgrade Codes do not match.

Table 12-44 • Validation Options (cont.)

Validation Option	Description
Match Product Code	Validates the Product Codes in both the base MSI and the transform. The transform is not applied if the Product Codes do not match.

Version comparison

Select the type of comparison you want to make.

Version level

Select whether you want only the major versions compared or both the major and minor versions.

Multi-Line String Value Dialog Box

The Multi-Line String Value dialog box lets you enter a multi-line registry value. This dialog box opens when you are adding or editing a multi-line string value for a registry key in the Registry view.

(Multi-line edit field)

In the multi-line edit field, enter a line for each null-delimited string or modify it. Right-click the grid to display the context menu or press the following keys in the grid below associated with the following actions.

Table 12-45 • Multi-line Edit Field Actions

Action	Shortcut Keys
Add String	INSERT
Rename String	F2
Delete String	DELETE
Move Up	UP ARROW
Move Down	DOWN ARROW

Click OK when you are finished entering a line for each null-delimited string. The strings are automatically concatenated.



Note • *Strings that contain only spaces are allowed, but strings cannot be empty or "[~]", which is the separator for the strings as they are stored in the project and displayed in the Destination computer's registry data pane. This separator is not included when the registry value is created on the target system.*

Select how you want to modify the registry value



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

For Windows Installer–based projects, you can choose from the following options to indicate where you want to add the value.

Table 12-46 • Options for Modifying the Registry Value

Option	Description
Append	Add to the end of the existing registry value string.
Prepend	Add to the front of the existing registry value string.
Replace	Replace the existing registry value string.

New Dependency Dialog Box

The New Dependency dialog box opens when you click the Add or Modify button on the Dependencies tab of the InstallShield Prerequisite Editor. This dialog box lets you specify other InstallShield prerequisites (.prq files) on which this prerequisite depends.

Table 12-47 • New Dependency Dialog Box Options

Dialog Option	Description
File	Type the path and name of the .prq file that is required for the current InstallShield prerequisite. To find the .prq file by browsing, click the ellipsis (...) button. InstallShield prerequisite files are stored in the following location by default: *InstallShield Program Files Folder***SetupPrerequisites** For more information about the locations where InstallShield prerequisites are stored, see Specifying the Directories that Contain InstallShield Prerequisites.

New File Dialog Box

The New File dialog box opens when you click the Add button on the Files to Include tab of the InstallShield Prerequisite Editor. This dialog box lets you specify a file for your InstallShield prerequisite.

Table 12-48 • New File Dialog Box Options

Option	Description
File	Type the path and name of the file. To find the file by browsing, click the ellipsis (...) button.
URL to file	If end users should be able to download the file from the Web, specify the uniform resource locator (URL) in this box. This URL is the same one that InstallShield uses when installation authors use the Redistributables view to download an InstallShield prerequisite from the Internet to their local machines. For example, if the URL for the file is http://www.mywebsite.com/Folder1/Notepad.exe, enter the following in this box: http://www.mywebsite.com/Folder1

New Project Dialog Box

This dialog box is displayed when you create a new project in InstallShield. In the dialog box, you can select a project type, name your project, and provide a location for the project's files. After you select a project type and click OK, your project opens in InstallShield.



Project • *If you select an installation project such as InstallScript, Basic MSI, or InstallScript MSI, the Project Assistant launches to help you create your project.*

Dialog Box Tabs

Table 12-49 • New Project Dialog Box Tabs

Tab	Description
Common	This tab displays some of the most-often-used project types.
InstallScript	This tab displays all of the project types that use the InstallShield installation engine, including the InstallScript installation project. If you previously used InstallShield Professional, you will be familiar with these project types.
Windows Installer	This tab includes projects that use the Microsoft Windows Installer engine, including the Basic MSI installation project.
All Types	This tab displays all of the project types available in InstallShield. It also includes any project templates that you previously saved, as well as any templates that are available in your repository.

Dialog Box Options

Table 12-50 • New Project Dialog Box Options

Option	Description
Project Name	Type a name for your project in this box. ***Project*** *• When you create a Merge Module project, the name that you specify in this box should have 35 characters or fewer. This is because the name that you enter is used with the GUID in the ModuleID field of the* ***ModuleSignature*** *table of your object or module file, and the limit for the name portion of the ModuleID field is a maximum of 35 characters.* *When you create a DIM project, the name that you specify in this box should have 35 characters or fewer.*
Project Language	Select the language that should be used for your installation project.
Location	Type a location or click Browse to navigate to a project location. To change the default project location displayed, change the Project Location path, which is specified on the File Locations tab of the Options dialog box.
Create the project in Project Name subfolder	Select this check box if you want InstallShield to create a subfolder with your project's name in your Projects location.

Operating Systems Dialog Box

This dialog lets you specify the operating systems with which a component (in an InstallScript MSI project) is associated.

Dialog Options

Operating Systems

Select the operating systems on which the component should be installed. Leaving all operating systems unselected causes the component to be installed regardless of the target operating system.

Options Dialog Box

The Options dialog box enables you to specify preferences for creating projects and working in InstallShield.



Task ***To access the Options dialog box:***

On the **Tools** menu, click **Options**.

The Options dialog box is organized into multiple task-related tabs:

- General tab
- File Locations tab
- Path Variables tab
- User Interface tab
- Preferences tab
- Updates tab
- .NET tab
- Files View tab
- File Extensions tab
- Prerequisites tab
- Source Control tab
- Directory tab
- Resource tab
- Validation tab
- Repository tab
- SQL Scripts tab
- Merge Modules tab
- Quality tab

General Tab

The General tab on the Options dialog box enables you to set general project options.

Table 12-51 • General Tab Settings

Setting	Project Types	Description
Enforce Setup Best Practices	Basic MSI, InstallScript MSI	Select this check box to indicate that you want InstallShield to monitor your installation design to help you comply with Setup Best Practices.
Help window on top	All	Select this check box to have the help window remain on top of the InstallShield interface. If you want the help window to fall to the background when focus is given to InstallShield, clear this check box.
Stop build process when first error is encountered	Basic MSI, InstallScript, InstallScript MSI	Select this check box to abort the build process when a build error occurs.
Automatically create ISSetAllUsers action	Basic MSI, InstallScript MSI	Select this check box to add the ISSetAllUsers action to your sequences when your project contains one or more records in the Upgrade table.

File Locations Tab

The File Locations tab on the Options dialog box enables you to set the default directories for where your project files are located and where all of your merge modules can be found.

Table 12-52 • File Locations Tab Settings

Setting	Project Type	Description
Project Location	All	The project location becomes your Favorite folder for new installation projects. All of your source and release files, such as your project (.ism) file, installation package (.msi file), and disk image files, are stored in subfolders of your Favorites location, currently [unknown]. You can enter a complete path or click the Browse button to navigate to the folder. By default, your project is stored in the following location: **C:\InstallShield 2016 Projects** ***Tip*** • *Changing the project location does not move existing folders and files in the previous project location. You must build a release with a new name or manually move the folders to change their location.*

Table 12-52 • File Locations Tab Settings (cont.)

Setting	Project Type	Description
Dialogs Location	Basic MSI	Enter the path where the Dialog Wizard should search for dialogs. Dialogs in this location appear in the list presented in the Dialog Wizard's Dialog Template panel. Separate additional paths with a comma.

Path Variables Tab

The Path Variables tab enables you to set user-level path variables options.

Table 12-53 • Path Variable Tab Settings

Setting	Project Types	Description
Always recommend path variables	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, QuickPatch, Suite/ Advanced UI	Select this option if you want InstallShield to automatically assign existing path variables to your files. For example, if you are adding a file to your installation project that is stored in your Program Files folder, InstallShield uses <ProgramFilesFolder> rather than a hard-coded path. Select the check box below this option to have InstallShield automatically create a path variable for you. When a new variable is created, it is titled <MYVAR#> (where # is a successive number). You can change the name of this variable in the Path Variables editor.
Don't recommend path variables	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, QuickPatch, Suite/ Advanced UI	Select this option if you want to use hard-coded paths for all of your file links. InstallShield does not suggest path variables for you if this option is selected.
Always display the Path Variable Recommendation dialog to me	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, QuickPatch, Suite/ Advanced UI	Select this option if you want the Path Variable Recommendation dialog box to be displayed every time you associate a file with your installation project.

User Interface Tab

The User Interface tab on the Options dialog box sets your user interface preferences. These settings are maintained across all projects that you create on this system.

Table 12-54 • User Interface Tab Settings

Setting	Project Type	Description
Always create new components and features	Basic MSI, InstallScript, InstallScript MSI	If you want InstallShield to automatically create components and features when needed, select this check box. If you do not select this check box, InstallShield prompts you to create components and features when necessary.
Always confirm feature, component, and release deletion	Basic MSI, InstallScript, InstallScript MSI	If you want InstallShield to display a confirmation dialog box each time that you delete a feature, component, or release, select this check box.
Always confirm script terminology conversion	Basic MSI, InstallScript, InstallScript MSI	If you want InstallShield to display a confirmation dialog box when you open a project that was created in InstallShield Professional, select this check box. The dialog box asks if you want to convert your script to use the InstallScript lexicon.
Automatically determine best feature or component	Basic MSI, InstallScript, InstallScript MSI	If you want InstallShield to automatically determine the feature or component that should contain the registry data that you configure in the Registry view, select this check box. If this check box is cleared, InstallShield displays a dialog box that you can use to specify the appropriate feature or component name when you create registry data for the All Application data option or a Feature option in the View Filter.
Show Welcome panel for IDE wizards	Basic MSI, InstallScript, InstallScript MSI	If you want InstallShield to display a Welcome panel every time you run a wizard, select this check box.
Warn about Web site deletion	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	If you want InstallShield to display a confirmation dialog box when you delete a Web site in the Internet Information Services view and the Web site contains one or more IIS applications or virtual directories, select this check box. The confirmation dialog box alerts you that the Web site's applications and virtual directories will be deleted from your project if you proceed with deleting the Web site.
Show InstallScript project types	InstallScript	This check box is available in the version of InstallShield that is included with AdminStudio. By default, this check box is cleared. If you want InstallShield to hide the InstallScript and InstallScript MSI project types on the New dialog box, select this check box.

Preferences Tab

The Preferences tab on the Options dialog box sets your program preferences. These settings are maintained across all installation projects you create on this system.

Table 12-55 • Preferences Tab Settings

Setting	Project Type	Description
Self-Registration	Basic MSI, InstallScript MSI	This section enables you to select the self-registration method that InstallShield should use when you indicate that a file is self-registering. Changing this setting applies only to future self-registration settings and does not affect files that were previously designated as self-registering. • **InstallShield Self-Registration Table (ISSelfReg)**—Select this option to use the InstallShield Self-Registration table for all files designated as self registering. • **Windows Installer Self-Registration Table (SelfReg)**—Select this option to use the Windows Installer Self-Registration table for all files designated as self registering. This section also enables you to select the default timing for COM extraction of self-registering files. This is the default setting used if a self-registering file is added via the Files and Folders view or through the Component Wizard. • **COM Extraction will occur during the build**—Select this option to perform extraction during the build. COM information will appear in the build log file. • **COM Extraction will occur immediately when the file is added**—Select this option to extract COM data right after you add a COM file. When you select this option, InstallShield populates the component's COM Registration Advanced Settings view.
Run Commands	Basic MSI, InstallScript MSI	If you want InstallShield to automatically uninstall your product before you debug your installation or before you rerun it by relaunching it from the Build menu, select the **Uninstall the product automatically before installing or debugging** check box.
Referential Integrity	Basic MSI, InstallScript, InstallScript MSI	Select the **Maintain referential integrity** check box if you want InstallShield to automatically maintain referential integrity when you are working in the Direct Editor. For example, the Control table has a Dialog_ column, meaning that the column is a foreign key to the Dialog table. If you rename the key in the Dialog table, the key in the Control table is also renamed. Also, if you delete the entire dialog, all of the controls should be deleted. This is true for any tables with this type of foreign key relationship.

Table 12-55 • Preferences Tab Settings (cont.)

Setting	Project Type	Description
Project Reload	All	Select the **Reload last opened project on startup** check box if you want InstallShield to automatically reload the most recently opened project when you launch InstallShield.

Updates Tab

The Updates tab on the Options dialog box enables you to configure how often FlexNet Connect checks for updates to InstallShield. It also lets you specify whether automatic update notification should be enabled by default for all new projects that you create in InstallShield.

Table 12-56 • Updates Tab Settings

Setting	Project Types	Description
Check for software updates	All	Select an option from this list to indicate how often you want InstallShield to check for software updates. ***Note*** • *In order to configure this option, you must be running InstallShield with administrative privileges. If you do not have administrative privileges, this option is disabled. To learn more, see Launching InstallShield with vs. Without Administrative Privileges.*
Enable automatic update notification in all new projects	Basic MSI, InstallScript MSI	Select this check box if automatic update notification should be enabled in all new projects that you create in InstallShield. You can override this automatic update notification setting for specific projects if necessary. ***Note*** • *FlexNet Connect cannot be used to deploy an upgrade for your installation if FlexNet Connect was not enabled in the original installation.* ***Project*** • *For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.*

.NET Tab

The .NET tab on the Options dialog box is where you specify preferences for .NET projects. It is also where you specify the location of the Regasm.exe and InstallUtilLib.dll files, which are utilities that are included with the .NET Framework. These utilities are used for COM interop and .NET custom actions.

Table 12-57 • .NET Tab Settings

Setting	Project Type	Description
Default .NET Scan at Build Component Setting	Basic MSI, InstallScript MSI	In this list, select how the .NET Scan at Build setting should be configured for new components. This applies to components that are automatically created when you add files to your project.
.NET Framework File Locations	Basic MSI, InstallScript MSI	**Regasm.exe** and **InstallUtilLib.dll** are utilities that are included with each version of the .NET Framework. Specify the path for the directory that contains the version of these files that you want to use at build time for releases that include .NET installer classes and COM interop. • **32-Bit Location**—Type the path or browse to the location of **Regasm.exe** and **InstallUtilLib.dll**. • **64-Bit Location**—This option is disabled if you are using InstallShield on a 32-bit system. If you are using InstallShield on a 64-bit system, type the path or browse to the location of **Regasm.exe** and **InstallUtilLib.dll**.  ***Note*** • *In order to configure these options, you must be running InstallShield with administrative privileges. If you do not have administrative privileges, these options are disabled. To learn more, see Launching InstallShield with vs. Without Administrative Privileges.*

Files View Tab

In the Files View tab on the Options dialog box, you can indicate preferences such as which columns you want to be displayed in several areas of InstallShield.

If you make any changes on this tab, close and reopen your project to see the changes take effect.

Table 12-58 • Files View Tab Settings

Setting	Project Type	Description
Size	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	To include a Size column in the following areas of InstallShield, select this check box: • Files and Folders view • Files subview in the Components view • Application Files page of the Project Assistant
Link To	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	To include a Link To column in the following areas of InstallShield, select this check box: • Files and Folders view • Files subview in the Components view • Application Files page of the Project Assistant
Modified Date	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	To include a Modified Date column in the following areas of InstallShield, select this check box: • Files and Folders view • Files subview in the Components view • Application Files page of the Project Assistant
Destination	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	To include a Destination column in the Application Files page of the Project Assistant, select this check box.
Version	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	To include a Version column in the following areas of InstallShield, select this check box: • Files and Folders view • Files subview in the Components view  ***Important*** • *Selecting the Version check box will slow the performance of the aforementioned views.*
File Key	Basic MSI, DIM, InstallScript MSI, Merge Module	To include a Key column in the Files subview in the Components view, select this check box.

Table 12-58 • Files View Tab Settings (cont.)

Setting	Project Type	Description
Maximum files to display	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	Specify the maximum number of files that you want to be displayed for a dynamic file link in the following areas of InstallShield: • Files and Folders view • Files subview in the Components view • Application Files page of the Project Assistant The default value is 1000. If the dynamic file link includes more than the specified number of files, the top entry in the list of files is called ***List Truncated***.
Clean up unused components	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	If you want InstallShield to remove unused components from your project automatically, select this check box. When this check box is selected, you delete all of the files in a component, and that component is not required elsewhere, that component is deleted automatically. Examples of scenarios in which a component is still required and will not be deleted from the project automatically are: • The component is selected in the **Destination computer's folders** pane in the Files and Folders view. • A folder would no longer be displayed in the Files and Folders view if the component were removed. • The component is selected for an item in another area of the product (for example, for XML file changes that are configured in the XML File Changes view). • In InstallScript projects, DefaultComponent under the Application Target Folder is never deleted automatically.

File Extensions Tab



Project • *The File Extensions tab is applicable to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The File Extensions tab on the Options dialog box is where you specify preferences for Portable Executable (PE) files. InstallShield refers to this list of PE files on several different occasions. For example:

- When you add a PE file to a folder in the **Destination computer's folders** pane in the Files and Folders view, InstallShield creates a new component for it and sets it as the component's key file.
- When you create components by using the best practice option in the Component Wizard, InstallShield creates a new component for PE files and sets each PE file as the key file of its component.
- When you use the best practice method of component creation for dynamic file links, InstallShield creates at build time a separate component for each PE file in the dynamically linked folder. Each PE file is the key file of its component.

The following setting is available on this tab:

Table 12-59 • File Extensions Tab Settings

Setting	Description
Portable Executable File Extensions	In this box, type the file extensions that you would like InstallShield to consider to be PE files. Separate extensions with a comma. The default entry for this box is the following: **EXE,DLL,OCX,VXD,CHM,HLP,TLB, AX**

Prerequisites Tab

The Prerequisites tab on the Options dialog box enables you to set preferences for InstallShield prerequisites.

Table 12-60 • Prerequisites Tab Settings

Setting	Project Type	Description
Prerequisite File Locations (Current User) and Prerequisite File Locations (All Users)	Basic MSI, InstallScript, InstallScript MSI	Specify the paths for the folders where you store InstallShield prerequisite files (.prq files). To specify more than one location, separate each path with a comma, as in the following example: **C:\Prerequisites,C:\My Files\Prerequisites** Note that you can use path variables in the path, as in the following example: **<ISProductFolder>\SetupPrerequisites,<ISProjectFolder>\MyCustomPrerequisites** The All Users option is available if you want to run a command-line build under a system account for which you cannot easily update the user settings. InstallShield provides additional ways for specifying the folders that contain InstallShield prerequisite files. For more information, see Specifying the Directories that Contain InstallShield Prerequisites. ***Note*** • *In order to configure the All Users option, you must be running InstallShield with administrative privileges. If you do not have administrative privileges, this option is disabled. To learn more, see Launching InstallShield with vs. Without Administrative Privileges.*

Source Control Tab

Edit the settings on the Source Control tab of the Options dialog box to control how InstallShield interacts with your source code control program. All of the settings on this tab apply to every project the current user opens on this system.

Project • *The following options apply to both Windows Installer—based and InstallScript-based projects unless otherwise noted.*

- ***Use dialog for checkout***—*When this check box is selected, a checkout dialog box is displayed when you check out a project through InstallShield. If this check box is cleared, InstallShield automatically checks out the file without a comment when you select the* ***Check out project when edited*** *check box or if, on the Project menu, you point to Source Control and then click Check Out.*

- ***Get latest version when opening the project***—*Select this check box to retrieve the latest version of your project from your source control program when you open the project in InstallShield. If you do not get the latest version, you might edit an older version in your working directory—if someone else modified the project file that exists in source control in the interim.*
- ***Check out project when edited***—*Select this check box to have InstallShield automatically check your project file out of your source control program whenever you modify it in InstallShield. If your project file is not checked out, it is likely read only and you cannot save changes to it until you check it out.*
- ***Add new projects to source control***—*Select this check box if you want new projects automatically added to source control upon creating them in InstallShield.*
- ***Use XML format when creating new Windows Installer-based projects***—*When this check box is selected, XML is the default file format for Windows Installer projects that you create using the New Project dialog box. If this check box is cleared, binary is the default file format setting for Windows Installer projects. You can change the project's file format at any time by changing the value for the Project File Format setting in the General Information view.*



Note • *When a project file format is converted to either XML or binary format, the project file extension remains .ism.*



Project • *The* ***Use XML Format when creating new Windows Installer-based projects*** *option applies only to Windows Installer–based projects.*

User name

Specify the name you use to log on to the source control project.

Advanced

Click this button to view the source control program's Properties dialog box and make advanced modifications.

Directory Tab

On the Directory tab of the Options dialog box, you can set various design-time directory options for Windows Installer and InstallScript MSI projects.

Table 12-61 • Directory Tab Settings

Setting	Project Type	Description
Show INSTALLDIR	Basic MSI, InstallScript MSI	Select this check box to display INSTALLDIR as a predefined folder at the root level of the folder structure. If the check box not cleared, INSTALLDIR appears in its location relative to other folders. In the Files and Folders view, it appears as a subfolder beneath the ProgramFilesFolder and the "Your Company Name" folder.

Table 12-61 • Directory Tab Settings (cont.)

Setting	Project Type	Description
Always show directory nodes with different IDs separately	Basic MSI, InstallScript MSI	In the Files and Folders view, the default behavior for displaying two directory identifiers with the same path is to display them as a single directory. To show different directory identifiers with the same path as two separate directories in the Files and Folders view, select this check box.
Clean up unused directories	Basic MSI, InstallScript MSI	Select this check box to remove unused user-defined directories from your installation project. For example, if you set a component's destination to {MYDIR}[INSTALLDIR]MYDIR1 and subsequently change the destination to [INSTALLDIR], MYDIR1 is automatically deleted from your project if it is not used elsewhere in your project.

Resource Tab

On the Resource tab of the Options dialog box, you can specify the location of the resource compiler and resource linker programs on your system. These programs are required in order to display modified dialogs in an InstallScript MSI installation project

These fields are automatically populated with the default file locations and command-line options for each program, but you can edit them if necessary.

Table 12-62 • Resource Tab Settings

Setting	Project Type	Description
Resource Compiler Settings	Basic MSI, InstallScript, InstallScript MSI	In this section, you can specify the location of a resource compiler and provide command line options. • **Resource compiler location**—The default resource compiler (**rc.exe**) location appears in this field. To edit the resource compiler location, type or browse to the location of a resource compiler that is currently installed on your system. • **Resource compiler command line options**—To edit the resource compiler command line options, type over the existing command line options in the field. The default command line option string is /r "%1".

Table 12-62 • Resource Tab Settings (cont.)

Setting	Project Type	Description
Resource Linker Settings	Basic MSI, InstallScript, InstallScript MSI	In this section, you can specify the location of a resource linker and provide command line options. • **Resource linker location**—The default resource linker (**link.exe**) location appears in this field. To edit the resource linker location, type or browse to the location of a resource linker that is currently installed on your system. • **Resource linker command line option**—To edit the resource linker command line options, type over the existing command line options in the field. The default command line option string is "%1" /DLL /NOENTRY /NODEFAULTLIB /MACHINE:iX86 /OUT:"%2".

Validation Tab

On the Validation tab of the Options dialog box, you can specify the type of validation you want InstallShield to perform after a successful build.

Table 12-63 • Validation Tab Settings

Setting	Project Type	Description
Perform Patching & Upgrading validation	Basic MSI, InstallScript MSI	Select this check box to perform validation on both patches and upgrades.
Perform validation using	Basic MSI, InstallScript MSI	To perform validation every time that you build an installation project, select this check box and then select the check box of one or more validation suites. You can also browse for a new validation module (.cub file).
Perform merge module validation using	Merge Module	To perform validation every time that you build a merge module, select this check box and then select the check box of one or more validation suites. You can also browse for a new validation module (.cub file).

Click the Customize button to specify which internal consistency evaluators (ICEs) should be used for a specific validation suite. For more information, see Specifying Which ICEs, ISICEs, ISVICEs and ISBPs Should Be Run During Validation.

Repository Tab



Edition • *The boxes on the Repository tab are available in the Premier edition of InstallShield only.*

If you want to set up a network repository or change its location, name, or description, you can do so through the Repository tab on the Options dialog box.

Table 12-64 • Repository Tab Settings

Setting	Description
Network Repository XML File	This is the path and file name of the .xml file that InstallShield creates for the network repository. InstallShield also automatically creates subfolders in the same directory as the .xml file; these subfolders contain items that are published to the network repository.
Name	This is the name of the network repository.
Description	This is a description of the network repository.

SQL Scripts Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

On the SQL Scripts tab of the Options dialog box, you can set one option for SQL script support.

Table 12-65 • SQL Script Tab Settings

Setting	Description
Generate unique Windows Installer properties for new connections	To share Windows Installer properties between any new database connections that you add to your project, clear this check box. To use different Windows Installer properties for any new connections that you add, select this check box. This check box is cleared by default. For more information, see Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties.

Merge Modules Tab

The Merge Modules tab on the Options dialog box enables you to set preferences for merge modules and merge module projects.

Table 12-66 • Merge Module Tab Settings

Setting	Project Type	Description
Merge Module Locations (Current User) and Merge Module Locations (All Users)	Basic MSI, InstallScript, InstallScript MSI	Enter the paths where you store merge modules (.msm files). Separate additional paths with a comma, as in the following example: **C:\MergeModules,C:\My Files\MergeModules** Note that you can use path variables in the path, as in the following example: **<ISProductFolder>\Modules\i386,<ISProjectFolder>\MyCustomModules** You can also use environment variables in the path. The All Users option is available if you want to run a command-line build under a system account for which you cannot easily update the user settings. The first path that you list is where InstallShield should copy a merge module after it is built. The file is copied only if you select **Copy to Modules folder** in the Merge Module Options panel of the Release Wizard, if you select the **Copy to Merge Modules folder** option in the Publish Merge Module setting on the Events tab in the Releases view, or if you run **ISCmdBld.exe** with the -e command-line option. InstallShield creates the folder if it does not exist. InstallShield provides additional ways for specifying the folders that contain merge modules. For more information, see Specifying the Directories that Contain Merge Modules. ***Note*** • *In order to configure the All Users option, you must be running InstallShield with administrative privileges. If you do not have administrative privileges, this option is disabled. To learn more, see Launching InstallShield with vs. Without Administrative Privileges.*
Merge Module File Search Behavior	Basic MSI, InstallScript MSI	When you add a file to your project via the Best Practices Component Wizard or the Files view (with Best Practices active), InstallShield searches the merge modules to see if there is a module that contains that file and notifies you of any matches. In this section, you can select options that InstallShield uses to determine whether the files match.

Quality Tab

The Quality tab on the Options dialog box provides the option to join the Customer Experience Improvement Program, which works on improving the quality, reliability, and improvement of software and services from Flexera Software.

Participation is not mandatory, but Flexera Software appreciates your input.

Other IIS Properties Dialog Box

When you select a Web site, application, or virtual directory in the Internet Information Services view, InstallShield displays a grid that contains many settings. The Other IIS Properties dialog box opens when you click the ellipsis button (...) in the Other IIS Properties setting in the Advanced area of the grid. This dialog box is where you specify values for IIS settings that are not displayed in the other areas of this view.

To customize the value of a property, click the property name, and then click the Change Value button. The Edit ISIISMetaData Value dialog box opens, enabling you to set the value as needed.



***Note** • The other IIS property settings apply to IIS 6 and earlier. IIS 7 ignores these settings.*

For help on specific settings, see "Metabase Property Reference" on the MSDN Web site.

Other Window Styles Dialog Box



***Project** • This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The Other Window Styles dialog box provides additional window style options for the dialog or control that is in the Dialogs view. This dialog box opens when you click the ellipsis button (...) in the Other Window Styles setting. The Other Window Styles setting is displayed when you click a dialog or a certain type of control in the Dialogs view.

The options that are available depend on the type of dialog control that is active when you open the Other Windows Style dialog box.

- Dialogs
- Bitmap, icon, and text area controls
- Button controls (check box, radio button group, push button, and group box)
- Combo box controls
- List view controls
- Edit field and list box controls
- Line controls

- Progress bar controls
- Selection tree controls

For more information about the values, refer to the MSDN Library.

Other Window Styles for Dialogs



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for a dialog. For additional information, see the MSDN Library.

Table 12-67 • Window Styles for Dialogs

Value	Description
WS_POPUP	Creates a pop-up window.
DS_NOIDLEMSG	Suppresses WM_ENTERIDLE messages that the system sends to the owner of a dialog box while the dialog box is displayed.
DS_CONTROL	Creates a dialog box that can function as a child window of another dialog box. This allows the user to tab among the control windows of a child dialog box and use its accelerator keys.
DS_SYSMODAL	This style is obsolete and is included for compatibility with 16-bit versions of Windows. If you specify this style, the system creates the dialog box with the WS_EX_TOPMOST style. This style does not prevent the user from accessing other windows on the desktop. Do not combine with the DS_CONTROL style.
DS_3DLOOK	Displays text in the dialog box in non-bold font and draws three-dimensional borders around control windows in the dialog box.
DS_CENTER	Centers the dialog box in the working area of the end user's screen.
DS_LOCALEDIT	Specifies that edit controls in the dialog box use memory in the applications data section. By default, all edit controls in dialog boxes use memory outside the applications data section.
DS_FIXEDSYS	Causes the dialog box to use the SYSTEM_FIXED_FONT instead of the default SYSTEM_FONT. This is a monospaced font compatible with the System font in 16-bit versions of Windows operating systems earlier than 3.0.
DS_CENTERMOUSE	Centers the cursor in the dialog box.

Table 12-67 • Window Styles for Dialogs (cont.)

Value	Description
DS_SETFONT	Indicates that the header of the dialog box template contains additional data specifying the font to use for text in the client area and controls of the dialog box. The font data begins on the WORD boundary that follows the title array. It specifies a 16-bit point size value and a Unicode font name string. If this style is not specified, the dialog box template does not include the font data.
DS_ABSALIGN	Indicates that the coordinates of the dialog box are screen coordinates.
DS_CONTEXTHELP	Displays a question mark in the title bar of the dialog box. When the end user clicks the question mark, the cursor changes to a question mark with a pointer. If the end user then clicks a control in the dialog box, the control receives a WM_HELP message. The control should pass the message to the dialog box procedure. The help application displays a pop-up window that typically contains help for the control. DS_CONTEXTHELP is only a placeholder. When the dialog box is created, the system checks for DS_CONTEXTHELP and—if it is there—adds WS_EX_CONTEXTHELP to the extended style of the dialog box. WS_EX_CONTEXTHELP cannot be used with the WS_MAXIMIZEBOX or WS_MINIMIZEBOX styles.
DS_MODALFRAME	Creates a dialog box with a modal dialog box frame that can be combined with a title bar and window menu by specifying the WS_CAPTION and WS_SYSMENU styles.
DS_NOFAILCREATE	Creates the dialog box even if errors occur—for example, if a child window cannot be created or if the system cannot create a special data segment for an edit control.
DS_SETFOREGROUND	Causes the system to use the **SetForegroundWindow** function to bring the dialog box to the foreground.

Other Window Styles for Bitmap, Icon, and Text Area Controls



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for bitmap, icon, and text area controls. For additional information, see the MSDN Library.

Table 12-68 • Window Style Options for Bitmap, Icon, and Text Area Controls

Value	Description
WS_GROUP	Specifies the first control in a group of controls. All controls defined with this style after the first control belong to the same group.
SS_CENTERIMAGE	Specifies that, if the bitmap or icon is smaller than the client area of the static control, the rest of the client area is filled with the color of the pixel in the top left corner of the bitmap or icon. If the static control contains a single line of text, the text is centered vertically in the client area of the control.
SS_NOTIFY	Sends the parent window STN_CLICKED, STN_DBLCLK, STN_DISABLE, and STN_ENABLE notification messages when the user clicks or double-clicks the control.
SS_RIGHTJUST	Specifies that the lower right corner of a static control with the SS_BITMAP or SS_ICON style is to remain fixed when the control is resized. Only the top and left sides are adjusted to accommodate a new bitmap or icon.
WS_VSCROLL	Creates a window with a vertical scroll bar.
SS_REALSIZECONTROL	Adjusts the bitmap to fit the size of the static control. Certain display settings on a target machine can affect the size of the control area used to render an image, such as the system font or DPI scaling. This style bit automatically resizes bitmaps to fit their controls. This style is recommended instead of SS_REALSIZEIMAGE to support high DPI machines. Otherwise, the image may be clipped or not fill the entire control at runtime on some machines.
SS_REALSIZEIMAGE	Prevents a static icon or bitmap control (that is, static controls that have the SS_ICON or SS_BITMAP style) from being resized as it is loaded or drawn. If the icon or bitmap is larger than the destination area, the image is clipped.
SS_ENDELLIPSIS	If the end of a string does not fit in the rectangle, it is truncated and ellipses are added. If a word that is not at the end of the string goes beyond the limits of the rectangle, it is truncated without ellipses. Compare with SS_PATHELLIPSIS and SS_WORDELLIPSIS. ***Note*** • *SS_ENDELLIPSIS, SS_PATHELLIPSIS, and SS_WORDELLIPSIS are mutually exclusive.*

Table 12-68 • Window Style Options for Bitmap, Icon, and Text Area Controls (cont.)

Value	Description
SS_PATHELLIPSIS	Replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle. If the string contains backslash (\) characters, SS_PATHELLIPSIS preserves as much as possible of the text after the last backslash. Compare with SS_ENDELLIPSIS and SS_WORDELLIPSIS. ***Note*** • *SS_ENDELLIPSIS, SS_PATHELLIPSIS, and SS_WORDELLIPSIS are mutually exclusive.*
SS_WORDELLIPSIS	Truncates any word that does not fit in the rectangle and adds ellipses. Compare with SS_ENDELLIPSIS and SS_PATHELLIPSIS. ***Note*** • *SS_ENDELLIPSIS, SS_PATHELLIPSIS, and SS_WORDELLIPSIS are mutually exclusive.*

Other Window Styles for Button Controls

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for button controls (check box, radio button group, push button, and group box controls). For additional information, see the MSDN Library.

Table 12-69 • Window Styles for Button Controls

Value	Description
WS_GROUP	Specifies the first control in a group of controls. All controls defined with this style after the first control belong to the same group.
BS_LEFTTEXT	Places text on the left side of the radio button or check box when combined with a radio button or check box style.
BS_CENTER	Centers text horizontally in the button rectangle.
BS_BOTTOM	Places text at the bottom of the button rectangle.
WS_VSCROLL	Creates a window with a vertical scroll bar.
BS_VCENTER	Places text in the middle (vertically) of the button rectangle.

Table 12-69 • Window Styles for Button Controls (cont.)

Value	Description
BS_PUSHLIKE	Makes a button—such as a check box, three-state check box, or radio button—look and act like a push button. The button looks raised when it is not pushed or selected, and sunken when it is pushed or checked.
BS_FLAT	Specifies that the button is two-dimensional. It does not use the default shading to create a three-dimensional image.
BS_NOTIFY	Enables a button to send BN_KILLFOCUS and BN_SETFOCUS notification messages to its parent window. Buttons send the BN_CLICKED notification message regardless of whether it has this style. To get BN_DBLCLK notification messages, the button must have the BS_RADIOBUTTON or BS_OWNERDRAW style.
BS_MULTILINE	Wraps the button text to multiple lines if the text string is too long to fit on a single line in the button rectangle.

Other Window Styles for Push Button Controls

The following values apply only to push button controls. They are mutually exclusive.

Table 12-70 • Window Styles for Push Button Controls

Value	Description
BS_OWNERDRAW	Creates an owner-drawn button. The owner window receives a WM_DRAWITEM message when a visual aspect of the button has changed. Do not combine the BS_OWNERDRAW style with any other button styles.
BS_USERBUTTON	Obsolete, but provided for compatibility with 16-bit versions of Windows. Win32-based applications should use BS_OWNERDRAW instead.

Other Window Styles for Check Box Controls

The following values apply only to check box controls. They are mutually exclusive.

Table 12-71 • Window Styles for Check Box Controls

Value	Description
BS_AUTOCHECKBOX	Creates a button that is the same as a check box, except that the check state automatically toggles between checked and cleared each time the user selects the check box.
BS_3STATE	Creates a button that is the same as a check box, except that the box can be grayed as well as checked or cleared. Use the grayed state to show that the state of the check box is not determined.

Table 12-71 • Window Styles for Check Box Controls (cont.)

Value	Description
BS_AUTO3STATE	Creates a button that is the same as a three-state check box, except that the box changes its state when the user selects it. The state cycles through checked, grayed, and cleared.

Other Window Styles for Combo Box Controls



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for combo box controls. For additional information, see the MSDN Library.

Table 12-72 • Window Styles for Combo Box Controls

Value	Description
WS_GROUP	Specifies the first control in a group of controls. All controls defined with this style after the first control belong to the same group.
WS_VSCROLL	Creates a window with a vertical scroll bar.
CBS_SIMPLE	Displays the list box at all times. The current selection in the list box is displayed in the edit control.
CBS_DROPDOWN	Similar to CBS_SIMPLE, except that the list box is not displayed unless the user selects an icon next to the edit control.
CBS_OEMCONVERT	Converts text entered in the combo box edit control from the Windows character set to the OEM character set and then back to the Windows set. This results in proper character conversion when the application calls the **CharToOem** function to convert a Windows string in the combo box to OEM characters. This style is useful for combo boxes that contain file names and applies only to combo boxes created with the CBS_SIMPLE or CBS_DROPDOWN style.
CBS_HASSTRINGS	Specifies that an owner-drawn combo box contains items consisting of strings. The combo box maintains the memory and address for the strings so the application can use the CB_GETLBTEXT message to retrieve the text for a particular item.
CBS_UPPERCASE	Converts to uppercase all text in both the selection field and the list.

Table 12-72 • Window Styles for Combo Box Controls (cont.)

Value	Description
CBS_AUTOHSCROLL	Automatically scrolls the text in an edit control to the right when the user types a character at the end of the line. If this style is not set, only text that fits within the rectangular boundary is allowed.
CBS_DISABLENOSCROLL	Shows a disabled vertical scroll bar in the list box when the box does not contain enough items to scroll. Without this style, the scroll bar is hidden when the list box does not contain enough items.
CBS_NOINTEGRALHEIGHT	Specifies that the size of the combo box is exactly the size specified by the application when it created the combo box. Normally, the system sizes a combo box so that it does not display partial items.
CBS_OWNERDRAWFIXED	Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are all the same height. The owner window receives a WM_MEASUREITEM message when the combo box is created and a WM_DRAWITEM message when a visual aspect of the combo box has changed.
CBS_OWNERDRAWVARIABLE	Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are variable in height. The owner window receives a WM_MEASUREITEM message for each item in the combo box when you create the combo box and a WM_DRAWITEM message when a visual aspect of the combo box has changed.

Other Window Styles for List View Controls



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for list view controls. For additional information, see the MSDN Library.

Table 12-73 • Window Styles for List View Controls

Value	Description
WS_GROUP	Specifies the first control in a group of controls. All controls defined with this style after the first control belong to the same group.
LVS_SINGLESEL	Enables only one item to be selected at a time. By default, multiple items can be selected.

Table 12-73 • Window Styles for List View Controls (cont.)

Value	Description
LVS_NOLABELWRAP	Displays item text on a single line in icon view. By default, item text might wrap in icon view.
LVS_AUTOARRANGE	Specifies that icons automatically remain arranged in icon view and small icon view.
LVS_SHOWSELALWAYS	Always shows the selection highlighted, even if the control is not activated.
LVS_SORTASCENDING	Sorts items based on item text in ascending order.
LVS_SORTDESCENDING	Sorts items based on item text in descending order.
LVS_EDITLABELS	Enables item text to be edited in place. The parent window must process the LVN_ENDLABELEDIT notification message.
LVS_OWNERDATA	Creates a virtual list view control.
LVS_NOSCROLL	Disables scrolling, so all items must be displayed within the client area.
LVS_NOSORTHEADER	Specifies that column headers do not work like buttons. This style is useful if clicking a column header in report view does not carry out any action, such as sorting.
LVS_OWNERDRAWFIXED	Enables the owner window to paint items in report view. The list view control sends a WM_DRAWITEM message to paint each item; it does not send separate messages for each subitem. The itemData member of the DRAWITEMSTRUCT structure contains the item data for the specified list view item.
LVS_NOCOLUMNHEADER	Specifies that no column header is displayed in report view, which is the default view.
LVS_SHAREIMAGELISTS	Specifies that the control does not destroy the image lists assigned to it when it is destroyed. This style enables the same image lists to be used with multiple list view controls.

Types



Note • *The following values are mutually exclusive.*

Table 12-74 • Type Values

Value	Description
LVS_ICON	Specifies icon view.
LVS_REPORT	Specifies report view.

Table 12-74 • Type Values (cont.)

Value	Description
LVS_LIST	Specifies list view.
LVS_SMALLICON	Specifies small icon view.

Align



Note • *The following values are mutually exclusive.*

Table 12-75 • Alignment Values

Value	Description
LVS_ALIGNTOP	Specifies that items are aligned with the top of the list view control in icon view and small icon view.
LVS_ALIGNLEFT	Specifies that items are left-aligned in icon view and small icon view.

Other Window Styles for Edit Field and List Box Controls



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for edit field and list box controls. For additional information, see the MSDN Library.

Table 12-76 • Window Styles for Edit Field and List Box Controls

Value	Description
WS_GROUP	Specifies the first control in a group of controls. All controls defined with this style after the first control belong to the same group.
WS_VSCROLL	Creates a window with a vertical scroll bar.
LBS_STANDARD	Creates a bordered list box.
LBS_NOTIFY	The parent window receives an input message whenever the end user clicks or double-clicks a string.

Table 12-76 • Window Styles for Edit Field and List Box Controls (cont.)

Value	Description
LBS_NODATA	Specifies a no-data list box. Specify this style when the count of items in the list box will exceed one thousand. A no-data list box must also have the LBS_OWNERDRAWFIXED style, but must not have the LBS_SORT or LBS_HASSTRINGS style. A no-data list box resembles an owner-drawn list box except that it contains no string or bitmap data for an item. Commands to add, insert, or delete an item always ignore any specified item data; requests to find a string within the list box always fail. The system sends the WM_DRAWITEM message to the owner window when an item must be drawn. The itemID member of the DRAWITEMSTRUCT structure passed with the WM_DRAWITEM message specifies the line number of the item to be drawn. A no-data list box does not send a WM_DELETEITEM message.
LBS_NOINTEGRALHEIGHT	Creates a list box that is precisely the size specified by the application when it created the list box.
LBS_NOREDRAW	List box is not updated when changes are made.
LBS_MULTIPLESEL	Toggles string selection each time the end user clicks or double-clicks the string.
LBS_HASSTRINGS	Specifies that a list box contains items consisting of strings. The list box maintains the memory and addresses for the strings so that the application can use the LB_GETTEXT message to retrieve the text for a particular item. By default, all list boxes except owner-drawn list boxes have this style. You can create an owner-drawn list box either with or without this style.
LBS_USETABSTOPS	Allows a list box to recognize and expand tab characters when drawing strings.
LBS_MULTICOLUMN	Creates a multicolumn list box that scrolls horizontally.
LBS_EXTENDEDSEL	Allows the end user to select multiple items using the SHIFT key and the mouse, or key combinations.
LBS_DISABLENOSCROLL	Displays a disabled vertical scroll bar when the list box does not contain enough items to scroll.
LBS_WANTKEYBOARDINPUT	Passes WM_VKEYTOITEM or WM_CHARTOITEM messages when the end user presses a key while the list box has focus. This allows an application to perform special processing on the keyboard input.
LBS_OWNERDRAWFIXED	Indicates that the owner of the list box is responsible for drawing the list box's contents. Items in the list box are the same height.
LBS_OWNERDRAWVARIABLE	Indicates that the owner of the list box is responsible for drawing the list box's contents. Items in the list box are of variable height.

Other Window Styles for Line Controls



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for line controls. For additional information, see the MSDN Library.

Table 12-77 • Window Styles for Line Controls

Value	Description
SS_SIMPLE	Specifies a simple rectangle and displays a single line of left-aligned text in the rectangle. The text line cannot be shortened or altered in any way. Also, if the control is disabled, the control does not gray its text.
SS_ETCHEDVERT	Draws the left and right edges of the static control using the EDGE_ETCHED edge style. For more information, see the **DrawEdge** Windows API function in the MSDN Library.
SS_ETCHEDFRAME	Draws the frame of the static control using the EDGE_ETCHED edge style. For more information, see the DrawEdge Windows API function in the MSDN Library.
SS_ETCHEDHORZ	Draws the top and bottom edges of the static control using the EDGE_ETCHED edge style. For more information, see the DrawEdge Windows API function in the MSDN Library.
SS_BLACKRECT	Specifies a rectangle filled with the current window frame color. This color is black in the default color scheme.
SS_BLACKFRAME	Specifies a box with a frame drawn in the same color as the window frames. This color is black in the default color scheme.
SS_WHITERECT	Specifies a rectangle filled with the current window background color. This color is white in the default color scheme.
SS_WHITEFRAME	Specifies a box with a frame drawn with the same color as the window background. This color is white in the default color scheme.
SS_GRAYRECT	Specifies a rectangle filled with the current screen background color. This color is gray in the default color scheme.
SS_GRAYFRAME	Specifies a box with a frame drawn with the same color as the screen background (desktop). This color is gray in the default color scheme.
WS_GROUP	Specifies the first control in a group of controls. All controls defined with this style after the first control belong to the same group.

Table 12-77 • Window Styles for Line Controls (cont.)

Value	Description
WS_VSCROLL	Creates a window with a vertical scroll bar.
SS_NOTIFY	Sends the parent window STN_CLICKED, STN_DBLCLK, STN_DISABLE, and STN_ENABLE notification messages when the user clicks or double-clicks the control.
SS_REALSIZECONTROL	Adjusts the bitmap to fit the size of the static control. Certain display settings on a target machine can affect the size of the control area used to render an image, such as the system font or DPI scaling. This style bit automatically resizes bitmaps to fit their controls. This style is recommended instead of SS_REALSIZEIMAGE to support high DPI machines. Otherwise, the image may be clipped or not fill the entire control at runtime on some machines.
SS_RIGHTJUST	Specifies that the lower right corner of a static control with the SS_BITMAP or SS_ICON style is to remain fixed when the control is resized. Only the top and left sides are adjusted to accommodate a new bitmap or icon.
SS_CENTERIMAGE	Specifies that, if the bitmap or icon is smaller than the client area of the static control, the rest of the client area is filled with the color of the pixel in the top left corner of the bitmap or icon. If the static control contains a single line of text, the text is centered vertically in the client area of the control.
SS_REALSIZEIMAGE	Prevents a static icon or bitmap control (that is, static controls that have the SS_ICON or SS_BITMAP style) from being resized as it is loaded or drawn. If the icon or bitmap is larger than the destination area, the image is clipped.
SS_ENDELLIPSIS	If the end of a string does not fit in the rectangle, it is truncated and ellipses are added. If a word that is not at the end of the string goes beyond the limits of the rectangle, it is truncated without ellipses. Compare with SS_PATHELLIPSIS and SS_WORDELLIPSIS.  ***Note*** • *SS_ENDELLIPSIS, SS_PATHELLIPSIS, and SS_WORDELLIPSIS are mutually exclusive.*
SS_PATHELLIPSIS	Replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle. If the string contains backslash (\) characters, SS_PATHELLIPSIS preserves as much as possible of the text after the last backslash. Compare with SS_ENDELLIPSIS and SS_WORDELLIPSIS.  ***Note*** • *SS_ENDELLIPSIS, SS_PATHELLIPSIS, and SS_WORDELLIPSIS are mutually exclusive.*

Table 12-77 • Window Styles for Line Controls (cont.)

Value	Description
SS_WORDELLIPSIS	Truncates any word that does not fit in the rectangle and adds ellipses. Compare with SS_ENDELLIPSIS and SS_PATHELLIPSIS. ***Note*** • *SS_ENDELLIPSIS, SS_PATHELLIPSIS, and SS_WORDELLIPSIS are mutually exclusive.*

Other Window Styles for Progress Bar Controls

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for progress bar controls. For additional information, see the MSDN Library.

Table 12-78 • Window Styles for Progress Bar Controls

Value	Description
WS_GROUP	Specifies the first control in a group of controls. All controls defined with this style after the first control belong to the same group.
PBS_VERTICAL	The progress bar displays progress status vertically, from bottom to top.

Other Window Styles for Selection Tree Controls

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The following window style options are available for selection tree controls. For additional information, see the MSDN Library.

Table 12-79 • Window Styles for Selection Tree Controls

Value	Description
WS_GROUP	Specifies the first control in a group of controls. All controls defined with this style after the first control belong to the same group.
TVS_EDITLABELS	Enables the user to edit the labels of tree view items.
TVS_HASBUTTONS	Displays plus (+) and minus (-) buttons next to parent items. The user taps the buttons to expand or collapse a parent items list of child items. To include buttons with items at the root of the tree view, you must also specify the TVS_LINESATROOT style.
TVS_DISABLEDRAGDROP	Prevents the tree view control from sending TVN_BEGINDRAG notification messages.
TVS_SHOWSELALWAYS	Uses the system highlight colors to draw the selected item.
TVS_FULLROWSELECT	Enables full-row selection in the tree view. The entire row of the selected item is highlighted, and clicking anywhere on an item's row causes it to be selected. This style cannot be used in conjunction with the TVS_HASLINES style.
TVS_HASLINES	Uses lines to show the hierarchy of items.
TVS_LINESATROOT	Uses lines to link items at the root of the tree view control. This value is ignored if the TVS_HASLINES control is not also specified.
TVS_RTLREADING	Causes text to be displayed from right-to-left (RTL). Usually, windows display text left-to-right (LTR). Windows can be mirrored to display languages such as Hebrew or Arabic that read RTL. Typically, tree-view text is displayed in the same direction as the text in its parent window. If TVS_RTLREADING is set, tree-view text reads in the opposite direction from the text in the parent window.
TVS_NOTOOLTIPS	Disables ToolTips.
TVS_CHECKBOXES	Enables items in a tree view control to be displayed as check boxes. This style uses item state images to produce the check box effect.
TVS_TRACKSELECT	Enables hot tracking in a tree-view control.
TVS_SINGLEEXPAND	Causes the item being selected to expand and the item being unselected to collapse upon selection in the tree view. If the mouse is used to single-click the selected item and that item is closed, it will be expanded. If the user holds down the Ctrl key while selecting an item, the item being unselected will not be collapsed.
TVS_INFOTIP	Obtains ToolTip information by sending the TVN_GETINFOTIP notification.

Table 12-79 • Window Styles for Selection Tree Controls (cont.)

Value	Description
TVS_NOSCROLL	Disables horizontal scrolling in the control. The control will not display any horizontal scroll bars.
TVS_NONEVENHEIGHT	Sets the height of the items to an odd height with the TVM_SETITEMHEIGHT message. By default, the height of items must be an even value.

Outputs Dialog Box

The Outputs dialog displays information about a project output group in the File System Editor. This dialog is available for InstallShield projects created in Microsoft Visual Studio.

Task ***To access the Outputs dialog, do one of the following:***

1. Select the **Outputs** property in the **Properties** window when a project output group is selected in the **File System Editor**.
2. In the **Files and Folders** view, right-click an item in the **Source computer's files** pane and click **Resolve Project Output**.
3. In the **Files and Folders** view, right-click an item in the **Destination computer's files** pane and click **Resolve Project Output**.

Note • *If multiple project output groups are selected, information is displayed only for the first group selected.*

Dialog Options

Target Name

Displays the file name for the selected project output group as it will be displayed on a target computer. This field is read only.

Source Path

Displays the path to the project output group files on the development computer. This field is read only.

Overwrite Dialog Box

This dialog box lets you specify whether the files of the active component (the component selected in the Setup Design or Components view) overwrite already-existing versions on the target system always, never, or conditionally based on date/time stamp or version number.

Dialog Options

list box

Select whether files on the target system are always overwritten, never overwritten, or conditionally overwritten based on date/time stamp or version number.

The following controls are enabled only if "Overwrite files BY VERSION" or "Overwrite files BY VERSION THEN DATE (if necessary)" is selected in the list *box*.



Note *• If a file on the target system has a version number and the file on your distribution media does not, or vice versa, the file with no version number is treated as if it had a lower version number.*

Version: Newer

A file on the target system is overwritten if the file on your distribution media has a higher version number.

If the file on your distribution media and the file on the target system have the same version number, or if neither file has a version number, and you selected Overwrite files BY VERSION THEN DATE (if necessary) from the list box, then whether the file on the target system is overwritten is determined by which Date/Time option you select.

If neither file has a version number and you selected Overwrite files BY VERSION from the list box, then the file on the target system is not overwritten.

Version: Newer or Same

A file on the target system is overwritten if the file on your distribution media has a higher or the same version number.

If neither file has a version number and you selected Overwrite files BY VERSION THEN DATE (if necessary) from the list box, then whether the file on the target system is overwritten is determined by which Data/Time option you select.

If neither file has a version number and you selected Overwrite files BY VERSION from the list box, then the file on the target system is not overwritten.

Version: Older

A file on the target system is overwritten if the file on your distribution media has a lower version number.

If the file on your distribution media and the file on the target system have the same version number, or if neither file has a version number, and you selected Overwrite files BY VERSION THEN DATE (if necessary) from the list box, then whether the file on the target system is overwritten is determined by which Date/Time option you select.

If neither file has a version number and you selected Overwrite files BY VERSION from the list box, then the file on the target system is not overwritten.

The following controls are enabled only if Overwrite files BY DATE or Overwrite files BY VERSION THEN DATE (if necessary) is selected in the list box.

Date/Time: Newer

A file on the target system is overwritten if the file on your distribution media has a more recent date and time stamp.

Date/Time: Newer or Same

A file on the target system is overwritten if the file on your distribution media has a more recent or the same date and time stamp.

Date/Time: Older

A file on the target system is overwritten if the file on your distribution media has a less recent date and time stamp.

Patch Sequence Dialog Box

The Patch Sequence dialog box opens when you are creating a patch sequence in the Patch Design view.

Table 12-80 • Patch Sequence Dialog Box Options

Option	Description
Family Name	Specify the name of the family of patches to which this patch belongs. Windows Installer 3.0 and later uses the patch family to compare small-update patches with all of the other patches within the same family and determine the order in which each of the patches should be applied to the target machine. To identify that a patch configuration belongs to multiple families, create a separate row on the Sequence tab for every patch family. You may want a patch to belong to multiple families, for example, if the patch can update more than one product.
Target	Type the product code GUID in this box if you want to associate the patch configuration with a specific product. Entry in this box is not required. To identify that a patch configuration should target multiple GUIDs, create a separate row on the Sequence tab for every GUID and family combination.
Sequence Number	Specify the sequence number for the patch configuration. This value indicates the placement of the patch configuration within sequence of patches that belong to the same patch family and the same target GUID (if identified).
Supersede previous versions	If you want the patch to be used instead of all of the patches that have a lower sequence value and that are in the same patch family, select this check box. Selecting this check box does not guarantee that the patch will always supersede all earlier patches, since the earlier patches might belong to multiple patch families. Note that a small-update patch cannot supersede a minor upgrade patch or a major upgrade patch, regardless of whether this check box is selected.

Path Variable Overrides Dialog Box



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The Path Variable Overrides dialog box opens when you click the ellipsis button (...) in the Path Variable Overrides setting. The Path Variable Overrides setting is on the Build tab for a release in the Releases view. Use this setting to override one or more path variables in your project at build time for the selected release.

The Path Variable Overrides dialog box shows a check box for the path variables, environment variables, and registry variables that are configured in the Path Variables view. Note that it does not include check boxes for any predefined path variables, or for path variables that are already overridden for the same release in the Releases view.

Select the check box for each path variable that you would like to override. Then click the OK button. Under the Path Variable Overrides setting, InstallShield adds a new setting for each path variable that you selected. Use those settings to specify the new values for the path variables.

At build time, InstallShield overrides the values that are set in the Path Variables view with the values that are configured for the release in the Releases view.

Path Variable Recommendation Dialog Box

If you have selected the **Always display the Path Variable Recommendation dialog to me** option on the Path Variables tab of the Options dialog box, InstallShield displays the Path Variable Recommendation dialog box every time you associate a new source folder with your installation project.

The dialog box is also displayed if you selected the **Always recommend path variables** option on the Path Variables tab of the Options dialog box and one of the following is true:

- InstallShield can recommend more than one path variable for a particular source folder.
- InstallShield cannot recommend a path variable based on an existing path variable and you have not selected the auto-create check box on the Path Variables tab.

Dialog Options

Use the following path variable-based folder representation for the source folder

This option provides suggested path variables or path variable combinations for you to use instead of an absolute path. For example, if you have a path variable called <MyFiles> that points to **C:\Work\Files** and you add a file to your installation from **C:\Work\Files\Images**, it is recommended that you use **<MyFiles>\Images** rather than the full hard-coded path **C:\Work\Files\Images**.

This option is available only if InstallShield can recommend a path variable for the source folder.

Create a new path variable

To create a path variable for the source folder in which your files are located, select the Create a new path variable option. Selecting this option creates a standard path variable that is mapped to the folder in which your new files are located. Enter the new variable path name in the Path Variable Name box.



Caution • *Do not enclose the path variable's name in angle brackets. InstallShield automatically adds the angle brackets when you click OK.*

Use the following absolute path

If you do not want to use a path variable to represent a particular link, select the **Use the following absolute path** option. The absolute path to your source folder appears below this button.

Permissions Dialog Boxes for Files and Directories



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Permissions Dialog Box

The Permissions dialog box lets you configure settings for securing files and folders for end users who run your product in a locked-down environment. You can assign permissions for a file or folder to specific groups and users. For example, you may assign Read, Write, and Delete permissions for a particular file to the Administrators group, but only Read permissions for all of the users in a different group.

Depending on what is selected for the Locked-Down Permissions setting in the General Information view of your project, InstallShield adds permissions data to either the **ISLockPermissions** table or the **LockPermissions** table. To learn more, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.

The following table describes the different areas on the Permissions dialog box.

Table 12-81 • Areas of the Permissions Dialog Box

Area	Description
Name(s)	In this grid, you can enter any combination of domain and user names. To add an entry, right-click the grid and click New. You can modify or delete entries using the same context menu. To specify the current user's domain, select **[%USERDOMAIN]** in the Domain field. To specify the user that is currently running the installation, select **[LogonUser]** in the User field. To set permissions for user accounts on a local system, leave the Domain field blank. Note that you can enter in the Domain or User fields any Windows Installer property that is set at run time—for example, **[MYPROPERTY]**. If the custom InstallShield handling option is selected for the Locked-Down Permissions setting in the General Information view, the User field contains a list of well-known security identifiers (SIDs). Most of the SIDs are not supported by the traditional Windows Installer handling option. The custom InstallShield handling option supports localized names for all of the SIDs that are listed in the User field. With the traditional option, if you try to use a localized name to set permissions on a non-English system, the installation may fail.  ***Tip*** • *For more information on the custom InstallShield handling option and the traditional Windows Installer handling option, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.*
Permissions	Select a name in the Name(s) area, and then select or clear the check boxes in the Permissions box to configure the corresponding permissions for the file or folder. Once you have selected a permission, you can click the Advanced button to specify other associated permissions and advanced settings.
Deny Access	If you want to explicitly deny the permissions that you are selecting in the Permissions box, select this check box. This check box is available only if the custom InstallShield handling option is selected for the Locked-Down Permissions setting in the General Information view. The traditional Windows Installer handling option does not include support for this behavior.  ***Tip*** • *For more information on the custom InstallShield handling option and the traditional Windows Installer handling option, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.*

Advanced Permissions Dialog Box

If you click the Advanced button on the Permissions dialog box, the Advanced Permissions dialog box opens. The following table describes the different areas on the Advanced Permissions dialog box.

Table 12-82 • Areas of the Advanced Permissions Dialog Box

Area	Description
Advanced Permissions	In this box, select the check boxes for the permissions that you want to set.
Apply these permissions to child objects	If you are configuring permissions for a folder and you want the permissions to be applied to all of the folder's subfolders and files, select this check box. This check box is available only if the custom InstallShield handling option is selected for the Locked-Down Permissions setting in the General Information view. The traditional Windows Installer handling option does not include support for this behavior. ***Tip*** • *For more information on the custom InstallShield handling option and the traditional Windows Installer handling option, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.*

Permissions Dialog Boxes for Registry Keys

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Permissions Dialog Box

The Permissions dialog box lets you configure settings for securing registry keys for end users who run your product in a locked-down environment. You can assign permissions for a registry key to specific groups and users. For example, you may assign Read, Write, and Delete permissions for a particular registry key to the Administrators group, but only Read permissions for all of the users in a different group.

Depending on what is selected for the Locked-Down Permissions setting in the General Information view of your project, InstallShield adds permissions data to either the **ISLockPermissions** table or the **LockPermissions** table. To learn more, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.

The following table describes the different areas on the Permissions dialog box.

Table 12-83 • Areas of the Permissions Dialog Box

Area	Description
Name(s)	In this grid, you can enter any combination of domain and user names. To add an entry, right-click the grid and click New. You can modify or delete entries using the same context menu. To specify the current user's domain, select **[%USERDOMAIN]** in the Domain field. To specify the user that is currently running the installation, select **[LogonUser]** in the User field. To set permissions for user accounts on a local system, leave the Domain field blank. Note that you can enter in the Domain or User fields any Windows Installer property that is set at run time—for example, **[MYPROPERTY]**. If the custom InstallShield handling option is selected for the Locked-Down Permissions setting in the General Information view, the User field contains a list of well-known security identifiers (SIDs). Most of the SIDs are not supported by the traditional Windows Installer handling option. The custom InstallShield handling option supports localized names for all of the SIDs that are listed in the User field. With the traditional option, if you try to use a localized name to set permissions on a non-English system, the installation may fail.  ***Tip*** • *For more information on the custom InstallShield handling option and the traditional Windows Installer handling option, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.*
Permissions	Select a name in the Name(s) area, and then select or clear the check boxes in the Permissions box to configure the corresponding permissions for the registry key. Once you have selected a permission, you can click the Advanced button to specify other associated permissions and advanced settings.
Deny Access	If you want to explicitly deny the permissions that you are selecting in the Permissions box, select this check box. This check box is available only if the custom InstallShield handling option is selected for the Locked-Down Permissions setting in the General Information view. The traditional Windows Installer handling option does not include support for this behavior.  ***Tip*** • *For more information on the custom InstallShield handling option and the traditional Windows Installer handling option, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.*

Advanced Permissions Dialog Box

If you click the Advanced button on the Permissions dialog box, the Advanced Permissions dialog box opens. The following table describes the different areas on the Advanced Permissions dialog box.

Table 12-84 • Areas of the Advanced Permissions Dialog Box

Area	Description
Advanced Permissions	In this box, select the check boxes for the permissions that you want to set.
Apply these permissions to child objects	If you are configuring permissions for a registry key and you want the permissions to be applied to all of the key's subkey, select this check box. This check box is available only if the custom InstallShield handling option is selected for the Locked-Down Permissions setting in the General Information view. The traditional Windows Installer handling option does not include support for this behavior. ***Tip*** • *For more information on the custom InstallShield handling option and the traditional Windows Installer handling option, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.*

Platform Suites Dialog Box

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The Platform Suites dialog box lets you specify the platform suites with which the selected component is associated. To access this dialog box, click the ellipsis button (...) in the Platform Suite(s) setting for a component in the Components view or the Setup Design view.

Table 12-85 • Platform Suites Dialog Box Settings

Setting	Description
Install without performing platform suite verification on the target system	The installation of the component's files does not depend on the target system's suite.
Install when the target system has one or more of the specified platform suite(s)	The installation installs the component's files only if at least one of the selected suites exists on the target system. If you select this option, select the check boxes of the appropriate platform suites.

Table 12-85 • Platform Suites Dialog Box Settings (cont.)

Setting	Description
Install only when the target system has all the specified platform suite(s)	The installation installs the component's files only if all of the selected suites exist on the target system. If you select this option, select the check boxes of the appropriate platform suites.
Show all platform suites	To see all of the available suites, select this check box. The full list are those that can be specified in the Windows API's OSVERSIONINFOEX data structure (as documented on MSDN). To see only the most common suites, clear this check box.

Platforms Dialog Box



Project • *Some of this information applies to InstallScript projects, and some of this information applies to InstallScript MSI projects.*

Depending on how you launch the Platforms dialog box, you can specify either of the following:

- The platforms that you want to be available when you to select operating system requirements for components or releases in your project. This is available for InstallScript projects.
- The platform requirements for a component. This is available for InstallScript and InstallScript MSI projects.

Platform Support at the Project Level (InstallScript Projects)

If you access the Platforms dialog box at the project level, you can specify the platforms that you want to be available when you to select operating system requirements for components or releases in your project.



Note • *Specifying platforms at the project level does not create target system requirements for running the installation. To create target system requirements in an InstallScript project, you can use the SYSINFO structure to identify the operating platform of the target system.*



Task ***To access the Platforms dialog box at the project level:***

1. In the View List under **Installation Information**, click **General Information**.
2. In the **Platform Filtering** setting, click the ellipsis button (**...**).

The following table shows the settings that are displayed on the Platforms dialog box when you are accessing it at the project level.

Table 12-86 • Platforms Dialog Box Settings for a Project

Setting	Description
The project supports all platforms that InstallShield supports	Select this option if you want InstallShield to list all of the supported run-time platforms for the following settings: • Operating Systems setting at the component level (InstallScript and InstallScript MSI projects) • Platform(s) setting at the release level (InstallScript projects) If you select this option, you can designate that a particular component or release in your project is targeted for any one or more supported platforms.
The project supports only the platforms that are selected below	Select this option if you want InstallShield to list only some platforms that are displayed for the following settings: • Operating Systems setting at the component level (InstallScript and InstallScript MSI projects) • Platform(s) setting at the release level (InstallScript projects) Then select the platforms that should be displayed.  ***Tip*** • *To select multiple consecutive operating systems, select the first file, press and hold SHIFT, and select the last operating systems. To select multiple nonconsecutive operating systems, select the first operating systems, press and hold CTRL, and select each additional operating systems.* In general, if a platform is not listed for this setting at the project level, you cannot designate that a particular component or release in your project is targeted for that platform.

Platform Support at the Component Level (InstallScript and InstallScript MSI Projects)

If you access the Platforms dialog box at the component level, you can specify which platforms are supported by the selected component.



Task ***To access the Platforms dialog box at the component level:***

1. In the View List under **Organization**, click **Components**.
2. In the **Components** explorer, click the component whose platform requirements you want to configure.
3. Click the value of the **Operating Systems** setting, and then click the ellipsis button (**...**).

The following table shows the settings that are displayed on the Platforms dialog box when you are accessing it at the component level.

Table 12-87 • Platforms Dialog Box Settings for a Component

Setting	Description
The files in this component should be installed when the setup is running on any platform	If the selected component is operating system independent—that is, if none of the component's data are specific to certain operating systems—select this option.
The files in this component should be installed only when the setup is running on one of the platforms checked below	If the selected component is specific to one or more operating systems, select this option and then select the appropriate operating systems. If the target machine's operating system does not match one of the operating systems that are specified for this setting, the component is not installed.  ***Tip*** • *To select multiple consecutive operating systems, select the first file, press and hold SHIFT, and select the last operating systems. To select multiple nonconsecutive operating systems, select the first operating systems, press and hold CTRL, and select each additional operating systems.*  ***Project*** • *For InstallScript projects, the list of operating systems that are displayed in this dialog box excludes any of the operating systems whose check box is cleared at the project level.* *Note that if the selected component supports platforms that are not supported by a release, InstallShield does not include the component in the release at build time.*

Postbuild Event Dialog Box



Edition • *This information applies to the Premier edition of InstallShield.*

The Postbuild Event dialog box opens when you click the ellipsis button (...) in the Postbuild Event setting. The Postbuild Event setting is on the Events tab for a release in the Releases view.

Use the Postbuild Event dialog box to specify one or more commands that you want to be run after InstallShield has built and signed the release.

To specify more than one command, enter each command on a separate line. At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one.

When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands.

For more information, see Specifying Commands that Run Before, During, and After Builds.

Prebuild Event Dialog Box



Edition • *This information applies to the Premier edition of InstallShield.*

The Prebuild Event dialog box opens when you click the ellipsis button (...) in the Prebuild Event setting. The Prebuild Event setting is on the Events tab for a release in the Releases view.

Use the Prebuild Event dialog box to specify one or more commands that you want to be run before InstallShield starts building the release. This event runs after InstallShield creates the release folder and log file, but before InstallShield starts building the release.

To specify more than one command, enter each command on a separate line. At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one.

When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands.

For more information, see Specifying Commands that Run Before, During, and After Builds.

Precompression Event Dialog Box



Edition • *This information applies to the Premier edition of InstallShield.*

The Precompression Event dialog box opens when you click the ellipsis button (...) in the Precompression Event setting. The Precompression Event setting is on the Events tab for a release in the Releases view.

Use the Precompression Event dialog box to specify one or more commands that you want to be run after InstallShield has built the .msi package and the .cab files (if your product's data files are to be stored in .cab files). Note that this event occurs after .cab files are streamed into the .msi package, but before the .msi package has been digitally signed and streamed into the **Setup.exe** file.

To specify more than one command, enter each command on a separate line. At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one.

When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands.

For more information, see Specifying Commands that Run Before, During, and After Builds.

Prerequisite Condition Dialog Box

The Prerequisite Condition dialog box opens when you click the Add button on the Conditions tab of the InstallShield Prerequisite Editor. It also opens when you select an existing condition on this tab and then click the Modify button.

The Prerequisite Condition dialog box lets you define installation conditions that determine whether an InstallShield prerequisite is already installed on the target machine. Failure to do so causes problems because the target system behaves as if the prerequisite was not properly installed. You can also create installation conditions that specify operating system, registry, or file requirements.

Table 12-88 • Prerequisite Condition Dialog Box Options

Option	Description
A registry key does or does not exist	If you select this option, type the name of the registry key and select the appropriate registry location to check on a 64-bit system: **32-bit** or **64-bit**. The **Default** radio button indicates you have not made a selection and 32-bit is used. If target machines that have this registry key should meet this installation condition, select **If the specified registry key exists**. If target machines that do not have this registry key should meet this installation condition, select **If the specified registry key does not exist**.
A registry entry has a specified value	If you select this option, type the name of the registry key, the name of the value, and the value data. Also, select the appropriate registry location to check on a 64-bit system: **32-bit** or **64-bit**. The **Default** radio button indicates you have not made a selection and 32-bit is used. In addition, select the option that should be used to compare the registry entry specified in this dialog box with the registry entry on the target machine. For example, if you specify **1.4.2** in the Value data box and select **is greater than**, target machines that have a value greater than 1.4.2 for this registry entry meet this installation condition.
A registry entry has a specified version value	If you select this option, type the name of the registry key, the name of the value, and the value data. The value data should be a version number. Note that all leading zeros for every field of the version number are ignored. For example, 3.5.3.0010 is greater than 3.5.3.009, and 3.5.3.009 is greater than 3.5.3.01. In addition, select the option that should be used to compare the registry entry specified in this dialog box with the registry entry on the target machine. Lastly, select the appropriate registry location to check on a 64-bit system: **32-bit** or **64-bit**. The **Default** radio button indicates you have not made a selection and 32-bit is used.

Table 12-88 • Prerequisite Condition Dialog Box Options (cont.)

Option	Description
A file does or does not exist	If you select this option, do one of the following: • Specify the path and name for the file. To specify a predefined folder, select the appropriate property in the list. The available properties are [CommonFiles64Folder], [CommonFilesFolder], [ProgramFiles64Folder], [ProgramFilesFolder], [System64Folder], [SystemFolder], and [WindowsFolder]. If you select one of the 64-bit folder locations, the installation checks the 64-folder location on 64-bit target systems and the 32-folder location on 32-bit target systems. • Specify just the file name—without the path or a property. The installation will search for the specified file in all directories defined for the PATH environment variable on the target machine. • Specify a registry key as a property. If you do this, the installation will resolve the registry key with the registry value. Note that the last element of the registry path is treated as a value name. If a backslash is the last character of the path, the value of the default registry entry for that registry key is used. For example, if you type the following as the value, the installation checks if the path specified as the value for the HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ORACLE_HOME registry entry contains a bin directory with a file named **oci.dll**: **[HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ORACLE_HOME]bin\oci.dll** If target machines that have this file should meet this installation condition, select **If the specified file is found in the location specified above**. If target machines that do not have this file should meet this installation condition, select **If the specified file is not found in the location specified above**.

Table 12-88 • Prerequisite Condition Dialog Box Options (cont.)

Option	Description
A file with a certain date exists	If you select this option, do one of the following: • Specify the path and name for the file. To specify a predefined folder, select the appropriate property in the list. The available properties are [CommonFiles64Folder], [CommonFilesFolder], [ProgramFiles64Folder], [ProgramFilesFolder], [System64Folder], [SystemFolder], and [WindowsFolder]. If you select one of the 64-bit folder locations, the installation checks the 64-folder location on 64-bit target systems and the 32-folder location on 32-bit target systems. • Specify just the file name—without the path or a property. The installation will search for the specified file in all directories defined for the PATH environment variable on the target machine. • Specify a registry key as a property. If you do this, the installation will resolve the registry key with the registry value. Note that the last element of the registry path is treated as a value name. If a backslash is the last character of the path, the value of the default registry entry for that registry key is used. For example, if you type the following as the value, the installation checks if the path specified as the value for the HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ORACLE_HOME registry entry contains a bin directory with a file named **oci.dll**: **[HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ORACLE_HOME]bin\oci.dll** In addition, type the date and—if appropriate—the time. For example: **4/26/2004 3:57:46 PM** Then select the appropriate match option to indicate when to run the InstallShield prerequisite.

Table 12-88 • Prerequisite Condition Dialog Box Options (cont.)

Option	Description
A file with a certain version exists	If you select this option, do one of the following: • Specify the path and name for the file. To specify a predefined folder, select the appropriate property in the list. The available properties are [CommonFiles64Folder], [CommonFilesFolder], [ProgramFiles64Folder], [ProgramFilesFolder], [System64Folder], [SystemFolder], and [WindowsFolder]. If you select one of the 64-bit folder locations, the installation checks the 64-folder location on 64-bit target systems and the 32-folder location on 32-bit target systems. • Specify just the file name—without the path or a property. The installation will search for the specified file in all directories defined for the PATH environment variable on the target machine. • Specify a registry key as a property. If you do this, the installation will resolve the registry key with the registry value. Note that the last element of the registry path is treated as a value name. If a backslash is the last character of the path, the value of the default registry entry for that registry key is used. For example, if you type the following as the value, the installation checks if the path specified as the value for the HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ORACLE_HOME registry entry contains a bin directory with a file named **oci.dll**: **[HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ORACLE_HOME]bin\oci.dll** In addition, type the version number of the file. For example: **6.7.8.9** Then select the appropriate match option to indicate when to run the InstallShield prerequisite.

Table 12-88 • Prerequisite Condition Dialog Box Options (cont.)

Option	Description
Setup is running on a specified platform	If you select this option, click one of the predefined platforms in the **Select which platform the prerequisite should be run on** box, or select Custom to define your own platform requirements. To specify a range of service pack numbers for which this prerequisite condition is true, use the Service Packs boxes. For example, if target system must have service pack 2, 3, or 4, enter 2 in the first box and 4 in the second box. To target all future service packs, leave the second box blank. To target only service pack 3, enter 3 in both boxes. If you select Custom to define your own operating system requirements, you can configure any one or more of the following settings in the Custom area: • **Platform ID**—Select the required platform. If the value that you select matches the dwPlatformId member of the OSVERSIONINFOEX structure of the target system's operating system, this part of the operating system condition evaluates as true. • **Major Version**—Specify the required major version of the operating system. If the value that you specify matches the dwMajorVersion member of the OSVERSIONINFOEX structure of the target system's operating system, this part of the operating system condition evaluates as true. If you type **0** in this box, the operating system condition that InstallShield creates does not include any major version or minor version requirements; the behavior is the same if you leave the Major Version and Minor Version boxes blank. • **Minor Version**—Specify the required minor version of the operating system. If the value that you specify matches the dwMinorVersion member of the OSVERSIONINFOEX structure of the target system's operating system, this part of the operating system condition evaluates as true. Note that **0** is a valid value for this box.

Table 12-88 • Prerequisite Condition Dialog Box Options (cont.)

Option	Description
Setup is running on a specified platform (cont.)	• **Product (OS) Type**—Select the required operating system type. If the value that you select matches the wProductType member of the OSVERSIONINFOEX structure of the target system's operating system, this part of the condition evaluates as true. Note that if the InstallShield prerequisite supports both servers and domain controllers, you can select the **Server or Domain Controller** option. • **Processor Architecture**—Select the required processor architecture (for example, 32-bit or 64-bit Windows). If the architecture that you select matches the architecture of the target system's operating system, this part of the operating system condition evaluates as true. • **Minimum CSD Version**—Specify the minimum required service pack or other version information for the operating system. Note that this setting is provided for compatibility with InstallShield prerequisites that were created with earlier versions of InstallShield, and to enable subversions of Windows 9x platforms to be distinguished. Using this setting to specify a particular service pack is not recommended; if you need to specify a particular service pack, use the Minimum Service Pack box. If the value that you specify is the same version as or an earlier version than the szCSDVersion member of the OSVERSIONINFOEX structure of the target system's operating system, this part of the operating system condition evaluates as true. For example, if you specify " A" and the szCSDVersion value on the target machine is " B", this part of the operating system condition evaluates as true. It evaluates as false if you specify " B" but the value on the target machine is " A". Similarly, if you specify "Service Pack 1" and the value on the target machine is "Service Pack 2", this part of the operating system condition evaluates as true. It evaluates as false if you specify "Service Pack 2" but the value on the target machine is "Service Pack 1". Note that to prevent comparison problems, any leading and trailing spaces in the version that you specify and in the value of the szCSDVersion member on the target system are ignored.
Setup is running on a specified platform (cont.)	• **Minimum Build No**—Specify the minimum required build number of the operating system. If the value that you specify is the same version as or an earlier version than the dwBuildNumber member of the OSVERSIONINFOEX structure of the target system's operating system, this part of the operating system condition evaluates as true. For more information on specifying an operating system condition, see Adding an Operating System Condition for an InstallShield Prerequisite. For details about the OSVERSIONINFOEX structure, see OSVERSIONINFOEX structure on the MSDN Web site.

Privileges Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Privileges dialog box lets you specify one or more privileges that are required by the service that you are configuring. This dialog box opens in the following scenario:

- You are configuring a service in the Services view, or in the Advanced Settings area for a component in the Setup Design view (installation projects only) or the Components view.
- The **Change list of required privileges** option is selected in the Configuration Type setting for an event in the Configure Settings category of settings in the right pane, and you click the ellipsis button (...) in the Arguments setting for that event.

Select one or more check boxes.

The change takes effect the next time that the system is started.

Product Condition Builder Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Product Condition Builder dialog box simplifies the process of creating install conditions by providing a selection of valid Windows Installer properties and conditional expression operators.

For example, if your product requires 64 MB of RAM in order to run properly, you can use the Product Condition Builder dialog box to create a condition. At run time, the Windows Installer checks the target system to determine how much RAM is installed. If it is less than 64 MB, the Windows Installer displays an error message and exits the installation.

Table 12-89 • Product Condition Builder Dialog Box Settings

Setting	Description
Conditions	This box displays a grid with the following columns: • **Condition**—This column is where you define the conditions. You can type a condition directly in this column, or you can use the Properties list, the Operators list, and the Add buttons to build a condition statement. • **Message**—This column is where you specify the error message that should be displayed if the condition evaluates to false. When you type a value for this column, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. To add a new condition to this box, click the New Condition button, and then enter the appropriate information in the Condition and Message columns. To delete a condition in this box, select the condition, and then click the Delete Conditions button.  ***Important*** • *When you click the OK button on this dialog box, InstallShield performs basic condition validation; however, you should still double-check that your condition statements evaluate to the expected outcome. For more information and example conditions, see Building Conditional Statements.* *Windows Installer dialogs, which display the text that you specify in the Message column, do not recognize the escape sequences \r (carriage return), \n (new line), or \t (tab).*
Properties	This list contains common Windows Installer properties, as well as any properties that have been added in the Property Manager view. Select the property that you want to evaluate, and then click the **Add** button next to this list. If the list does not contain the property that you want to evaluate, you can type the appropriate property name in the Condition column.
Operators	This list contains all of the valid operators that are recognized by the Windows Installer. Select the operator that you want to use and then click the Add button next to this list. InstallShield appends the operator to the conditional statement in the Conditions box.

Project Settings Dialog Box

The Project Settings dialog box lets you modify information about the open project.

This dialog box is displayed with the following tabs when you select the Project menu's Settings command. This dialog box is displayed with only the last two of the following tabs when you right-click an item in the Objects view's "InstallShield Objects/Merge Modules" pane and select Register new object or Properties. In the last case, all properties are uneditable.

- Project Tab
- Application Tab (in InstallScript Projects) (in Windows Installer–based projects)
- Application Tab (in InstallScript Projects) (in InstallScript and InstallScript Object projects)
- Platforms Tab (in InstallScript and InstallScript Object projects)
- Languages Tab
- Maintenance Tab (in InstallScript projects)
- Language-Independent Object Properties Tab (in InstallScript Object projects)
- Language-Specific Object Properties Tab (in InstallScript Object projects)

Project Tab

The Project tab on the Settings dialog box displays basic information about the open project. This dialog box opens when you select the Project menu's Settings command.

Author

(Optional) Enter the name of the project author.

Comments

(Optional) Enter any comments about the project.

Application Tab (in Windows Installer–Based Projects)



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The Application tab on the Settings dialog box lets you view and modify information about the product that is installed by the open project. This dialog box opens when you click the Settings command on the Project menu.

Table 12-90 • Application Tab Settings

Setting	Description
Product Code	This setting displays a GUID that uniquely identifies this product. To have InstallShield generate a different GUID for you, click the Change button next to this setting. Since this code uniquely identifies your product, changing the product code after you have already distributed your release is not recommended. For more information, see Setting the Product Code in a Windows Installer–Based Project.
Upgrade Code	This setting displays the GUID that can be used for your product's upgrade code. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. The upgrade code is a GUID that identifies a related set of products. The Windows Installer uses a product's upgrade code when performing major upgrades of an installed product. The upgrade code, stored in the **UpgradeCode** property, should remain the same for all versions of a product. For more information, see Setting the Upgrade Code.
Product Name	Enter the name of the product. For information on how the product name is used, see Specifying a Product Name.
Product Version	Enter the version number for your product. The version number must contain only numbers, and it must be in the format *aaa.bbb.ccccc*, where *aaa* represents the major version number, *bbb* represents the minor version number, and *ccccc* represents the build number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* is 65,535. For more information, see Specifying the Product Version.
Application Type	Select the type of application from the list or type the name of an application type if it is not listed. This information is stored in the project file and is for your reference only. It is never displayed to the end user.



Tip • *You can also configure the aforementioned settings in the General Information view.*

Application Tab (in InstallScript Projects)



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The Application tab on the Settings dialog box lets you view and modify information about the project that is installed by the open project. This dialog box opens when you click the Settings command on the Project menu.

Table 12-91 • Application Tab Settings

Setting	Description
Product Code	Enter a GUID that uniquely identifies this product. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. The product code is used to associate uninstallation or maintenance with the original installation. A new GUID is automatically generated for each new project that you create, including copies of existing projects. Once you have changed a project's product code, its previous GUID cannot be recovered. For these reasons, changing a project's product code is typically not necessary and should be approached with caution. For more information, see Setting the Product Code in an InstallScript-Based Project.
Product Name	Enter the name of the product. For information on how the product name is used, see Specifying a Product Name.  ***Tip*** • *Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.* *The product name is stored in the InstallScript system variable IFX_PRODUCT_NAME.*

Table 12-91 • Application Tab Settings

Setting	Description
Product Version	Enter the version number for your product. The version number must contain only numbers, and it must be in the format *aaa.bbb.ccccc*, where *aaa* represents the major version number, *bbb* represents the minor version number, and *ccccc* represents the build number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* is 65,535. For more information, see Specifying the Product Version.  ***Tip*** • *Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.*
Company Name	Enter the name of your company. This value is used in the default script to set TARGETDIR (if the string entry COMPANY_NAME does not exist); it can be retrieved at run time by calling the **MediaGetData** function.  ***Tip*** • *Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.*
Executable File	Enter the name of the application's main executable file. This value can be retrieved at run time by calling the **MediaGetData** function.  ***Tip*** • *Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.*
Application Type	Select the type of application from the list or type the name of an application type if it is not listed. This information is stored in the project file and is for your reference only. It is never displayed to the end user.
URL	Enter a product URL. This information is stored in the project file and is for your reference only. It is never displayed to the end user.  ***Tip*** • *Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.*



Tip • *You can also configure the aforementioned settings in the General Information view.*



Caution • *You must specify a non-null value in the Product Version box; otherwise, build error -7044 occurs.*

Platforms Tab



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The Platforms tab on the Settings dialog box is where you specify the platforms that you want to be available when you to select operating system requirements for components or releases in your project.



Note • *Specifying platforms at the project level does not create target system requirements for running the installation. To create target system requirements in an InstallScript project, you can use the SYSINFO structure to identify the operating platform of the target system.*

The following table shows the settings that are displayed in the Platforms tab.

Table 12-92 • Platforms Dialog Box Settings for a Project

Setting	Description
The project supports all platforms that InstallShield supports	Select this option if you want InstallShield to list all of the supported run-time platforms for the following settings: • Operating Systems setting at the component level (InstallScript and InstallScript MSI projects) • Platform(s) setting at the release level (InstallScript projects) If you select this option, you can designate that a particular component or release in your project is targeted for any one or more supported platforms.

Table 12-92 • Platforms Dialog Box Settings for a Project (cont.)

Setting	Description
The project supports only the platforms that are selected below	Select this option if you want InstallShield to list only some platforms that are displayed for the following settings: • Operating Systems setting at the component level (InstallScript and InstallScript MSI projects) • Platform(s) setting at the release level (InstallScript projects) Then select the platforms that should be displayed.  ***Tip*** • *To select multiple consecutive operating systems, select the first file, press and hold SHIFT, and select the last operating systems. To select multiple nonconsecutive operating systems, select the first operating systems, press and hold CTRL, and select each additional operating systems.* In general, if a platform is not listed for this setting at the project level, you cannot designate that a particular component or release in your project is targeted for that platform.

Languages Tab

The Languages tab on the Settings dialog box lets you view and modify the list of languages included in the open project. This dialog box opens when you click the Settings command on the Project menu.

check boxes

A checked check box next to a language indicates that the language is included in the project; an unchecked check box indicates that the language is not included. To add or remove a language from the project, click its check box to toggle it to the desired state.

Removing a language from the project also removes it from any components or releases that had been using that language.

Maintenance Tab



Project • *This information applies to InstallScript projects.*

The Maintenance tab on the Settings dialog box lets you select an option for the behavior of your setup when your end user reruns it. This dialog box opens when you click the Settings command on the Project menu.

The Multi-Instance option lets your end users run a setup multiple times as a first setup, rather than a maintenance setup. For more information on these options, see Running an InstallScript Installation Multiple Times.

Language-Independent Object Properties Tab



Project • *This information applies to InstallScript Object projects.*

The Language-Independent Object Properties tab contains basic information about your object, such as the name of the object, the company that created it, and the location of the object. This dialog box opens when you select the Project menu's Settings command in an InstallScript object project. This dialog box also opens in an InstallScript project when you register an object from within the Objects view, or you right-click an object in the Objects view and click Properties. In the last case, all properties are uneditable.

Object Wizard

This option allows you to specify the type of wizard, if any, that you would like to use for your object's default language. The following wizard choices are available:

- **No Wizard**—Select this option if your object does not require a wizard.
- **Use InstallShield Object Stock Wizard**—Select this option to have InstallShield create a wizard for you based on the properties you created for your object. All read-only properties will be displayed but will not be editable. All read/write properties will be changeable by the user. Write-only properties will not be displayed.

  Note • *The InstallShield stock wizard does not support array properties. If your object's properties require array properties, you will need to create your own wizard.*

- **Use My Custom Wizard**—Select this option to use a wizard you created. InstallShield allows you to use a wizard created with either Visual C++ or Visual Basic. To use a wizard created with one of these two programs, enter the path to the DLL you created, or click the browse (...) button to navigate to that file.

The following controls are not displayed when you select the Project menu's Settings command.

Company

Enter the name of the company responsible for creating this object.

Version

Enter the version of this object.

Media File

Enter the path to the object's media file (Data1.hdr file), or click the browse button to navigate to that file.

Language-Specific Object Properties Tab



Project • *This information applies to InstallScript Object projects.*

The Language-Specific Object Properties tab contains basic information about your object, such as the name of the object, the company that created it, and the location of the object. This dialog box opens when you select the Project menu's Settings command in an InstallScript object project. This dialog box also opens in an InstallScript project when you register an object from within the Objects view, or you right-click an object in the Objects view and click Properties. In the last case, all properties are uneditable.

IDE Language

Select the language for which you want to specify object information. Available options are English and Japanese.

Use Default

Disabled if the default language, English, is selected. Check this box to specify that the object, when displayed in the selected language, use the same information that you specified for the default language.

The following controls are enabled only if the **Use Default** *check box is unchecked:*

Display Name

Enter the name of your object as you would like it to appear in the object gallery.

Short Name

Enter a shortened version of your object's name, if necessary. You can enter the same name as entered in the Display Name field.

HTML Help

Enter the path to the help file for this object, or click the browse (...) button to navigate to this file. This help file must consist of a single .htm file.

Icon File

Enter the path to the icon file for your object, or click the browse (...) button to navigate to this file. The icon you choose will be displayed next to your object in the object gallery. The icon you use for your object needs to be sized to 16x16 with a maximum of 16 colors.

Rename Key Dialog Box

The Rename Key dialog box is displayed if you click Edit after selecting Rename on the Resolve Conflict dialog box. The Rename Key dialog box displays a setting in which you can provide a new key value for the exported dialog, component, or custom action. When you rename the exported item, you avoid the conflict, assuming that no objects with the new name exist in the target project file.

Settings

Table 12-93 • Rename Key Dialog Box Settings

Setting	Description
Key Value	Type a new key value or name for the item in this field and press OK to rename the item.

Required Features Dialog Box



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The Required Features dialog box lets you specify features that must be installed if the current feature is installed. This dialog box is launched when you click the ellipsis button (...) in the Required Features setting for a feature in the Setup Design view or the Features view.

The Required Features dialog box has a Required Features for *SelectedFeatureName* box that shows all of the features in your project. Select the check box for each feature that the current feature requires.



Project • *The check box for the current feature is automatically selected in InstallScript MSI projects. It is automatically cleared in InstallScript and InstallScript Object projects; in either case, the selection status cannot be changed, and it does not affect whether the feature is installed.*

Resolve Conflict Dialog Box

The Resolve Conflict dialog box identifies a conflict when you are exporting a dialog, component, or custom action to another project, or when you are importing a DIM into an installation project. A conflict occurs when the target project has an item with the same name as but different values in its settings than the one you are exporting. Resolve the conflict by selecting one of the options and clicking OK, or by renaming the item in a key column.

If you click Cancel, the item is not exported to another project, or the DIM is not imported into the installation project.

Dialog Box Settings

Table 12-94 • Resolve Conflict Dialog Box Settings

Setting	Description
Table	The table shows all of the data that pertains to the dialog, control, component, custom action, or other item. If you are exporting an item to a different project, the Source Data column shows the values of the settings for the item that you are exporting from the current project to a different project. The Target Data column shows the values of the settings of the existing item in the project to which you are exporting. If you are importing a DIM project into an installation project, the Source Data column shows the values of the settings for the item in the DIM project. The Target Data column shows the values of the settings of the setting for the item in the installation project. Key icons identify key columns in the record. If there is a conflict between the values, that row appears in red. Your selection for resolving the conflict determines whether InstallShield overwrites the existing item and its settings.
Resolution	Indicate how you want to resolve the conflict. Available options are: • **Skip**—Use the value in the Source Data column. If additional conflicts remain, show the next conflict. • **Skip All**— Use the value in the Source Data column for the current conflict as well as any remaining conflicts. • **Overwrite**—Use the value in the Target Data column. If additional conflicts remain, show the next conflict. • **Overwrite All**—Use the value in the Target Data column for the current conflict as well as any remaining conflicts. • **Rename**—Use a different value for the key column so that there is no conflict for the item. To use this option, first select the row that contains the key that you want to rename, select the Rename option, and then click the Edit button. The Rename Key dialog box opens, letting you specify the new value for the key column.

Save As Dialog Box

On the File menu, click Save As to open the Save As dialog box. This lets you save a copy of the open project as a template or as a new project with a different name and location.

Dialog Box Settings

The Save As dialog box is a standard Windows dialog box. To display detailed help for any control except the following ones, select the control and press F1.

Table 12-95 • InstallShield-Specific Settings on the Save As Dialog Box

Setting	Description
Save as type	***Project*** *• The* ***Save as type*** *option is available for the following project types:* • *Basic MSI* • *DIM* • *InstallScript* • *InstallScript MSI* • *Merge Module* The **Save as type** list enables you to specify the type of file that you are saving. To create a template from the current project, select InstallShield Template (.ist).
Create 'Project Name' subfolder and save the project in the created folder	This check box controls where the project file is located on your system. To create a <Project Name> subfolder in the location specified in the Project Location box (on the File Locations tab of the Options dialog box) and to save the project in that subfolder, select this check box. For projects that contain InstallScript, the Script Files folder is also created in this subfolder. To save the .ism project in the location specified in the Project Location box, clear this check box.
Create and assign a new project GUID to the saved project	If the new project's GUID should be different than the GUID of the original project, select this check box. If the new project's GUID should be the same GUID of the original project, clear this check box.
Update the project settings appropriately based on the new project name	To use the name that you specified in the **File name** box as the value for the Project Name setting in the General Information view of the new project, select this check box. If the value of the new project's Name setting should be the same as that of the original project, clear this check box.

Script Conversion Dialog Box

InstallShield uses different terminology for InstallScript function names. For example, InstallShield uses the term *feature* where InstallShield Professional uses *component*, and vice versa. See Script Changes: Lexicon Conversion for more information.

When you open an .ipr file in InstallShield, this dialog allows you to indicate whether you want InstallShield to automatically update your script (.rul) file to use the InstallShield lexicon.

Note *• The InstallShield Help Library and other documentation uses the InstallShield script terminology.*

Script Editor Properties Dialog Box

The Script Editor Properties dialog box is where you specify how script code should be displayed in various views throughout InstallShield. This dialog box is available in views that enables you to edit a script file (for example, the InstallScript, SQL Scripts, and Custom Actions and Sequences views).

To open the Script Editor Properties dialog box, right-click in a script editor pane and then click Properties.



***Tip** • Note that the script editor properties are global per-user settings that affect all InstallShield script editors. For example, the same font is used for all script editors. If you change the font in the SQL Scripts view, the font is also changed in the InstallScript view the next time that the InstallScript is reloaded in that view.*

The settings on the Script Editor Properties dialog box are organized into the following main categories:

- General
- Colors

General Settings

The following settings are available in the General area on the Script Editor Properties dialog box.

Table 12-96 • General Settings on the Script Editor Properties DIalog Box

Setting	Description
Font	This setting displays the font, style, and size that is used for the content in the script editor. To change the option that is selected, click the ellipsis button in this setting. Doing so opens the Font dialog box, which enables you to view and modify the font, style, and size of script content.
Syntax highlighting	Specify whether you want to InstallShield to use color coding for various script elements that are displayed in all of the script editors. If you select Yes, InstallShield uses color coding according to the selections that are made for the settings in the Colors area of the Script Editor Properties dialog box. For more information, see Changing Colors for Syntax Highlighting in the Script Editors.
Auto completion	Specify whether you want InstallShield to display a pop-up list of alphabetically ordered functions, keywords, constants, and other script words that begin with the letters that you are typing. Auto completion also displays a list of available string identifiers when you enter the string constant operator (@) in the InstallScript view. The default value is Yes. Auto completion can increase your efficiency because it can reduce the time that you spend typing code. It can also help you avoid typographical errors in your code. For more information, see Using Auto Completion when Writing Code in the Script Editors.

Table 12-96 • General Settings on the Script Editor Properties DIalog Box (cont.)

Setting	Description
Include local variables	Specify whether you want InstallShield to include in the pop-up list all of the local variables that are defined in the selected script. The default value is Yes. Note that if you select No, the Functions, Properties, and Methods folders in the center pane of the InstallScript view does not list any functions, properties, or methods from your script files. If Yes is selected for this setting and you notice performance issues when you are typing code in the InstallScript view, you might want to change this setting to No. This setting is ignored if you select No for the **Auto completion** setting. For more information, see Using Auto Completion when Writing Code in the Script Editors.
Show function call tips	Specify whether you want InstallShield to display a function call tip—a type of tooltip—when you are typing a function call in your script. The function call tip shows the function's parameter information. It also shows a description of the function, as well as a description of the parameter that you are entering. The default value is Yes. For more information, see Viewing Function Call Tips for an InstallScript Function in the Script Editor.
Line numbering	Specify whether you want the left margin of the script editor to contain line numbers. The default value is No. For more information, see Going to a Line Number in a Script that Is Displayed in a Script Editor.
Syntax folding	Specify whether you want to use syntax folding for your script. If you select Yes, InstallShield adds a plus sign (+) or a minus sign (–) in the margin next to each line of code that starts an expandable or collapsible block of script. Click the plus sign to expand the hidden code. Click the minus sign to hide the code. The default value is No. Syntax folding can help you minimize the clutter of large scripts and focus on the code that is relevant to the work that you are currently doing. It can also help you see the overall structure of a script.
Automatic indentation	Specify whether you want InstallShield to indent new lines of code in your script. The default value is Yes. For more information, see Enabling or Disabling Automatic Indentation in the Script Editors.

Table 12-96 • General Settings on the Script Editor Properties DIalog Box (cont.)

Setting	Description
Show whitespace	Specify whether you want InstallShield to display a symbol or other mark in place of each instance of white space in your script. For example, if you select Yes, InstallShield displays each space character in the script as a middle dot (·) and each tab character as an arrow. InstallShield also identifies line breaks when whitespace is enabled. The default value is No
Tab width	Specify the tab size as a multiple of the character space size. For example, setting a tab space of 5 causes the cursor to move five character spaces when you press the TAB key.

Colors Settings

The Colors area on the Script Editor Properties dialog box is where you view and modify the foreground color and—in some cases—the background color for various script elements.

The following settings are available in the Colors area on the Script Editor Properties dialog box.

Table 12-97 • Color Settings on the Script Editor Properties DIalog Box

Setting	Description
Default	Specify the colors for the text that is not one of the predefined types of script words.
Selection	Specify the colors for text that is selected in the script editor.
Cursor	Specify the colors for the blinking cursor.
Keywords	Specify the colors for script keywords such as built-in functions, data types, and constants.
Identifiers	Specify the colors that are used to show identifiers. This includes variables.
Operators	Specify the colors that are used to show operators. Examples include parentheses, brackets, and braces; arithmetic symbols like plus, minus, asterisk, and slash; and comma.
Comments	Specify the colors for the comments in your code; this includes the comment delimiters such as /*, */, or //.
Strings	Specify the colors for strings; this includes the surrounding quotes.
Unterminated strings	Specify color overrides for strings that have not been closed with a terminating quote; this helps you find strings that could cause compilation errors.

Table 12-97 • Color Settings on the Script Editor Properties DIalog Box (cont.)

Setting	Description
Preprocessor directives	Specify the colors that are used to show preprocessor directives; this includes any lines that start with a pound sign (#).
Constants	Specify the colors that are used to show script constants.
Labels	Specify the colors that are used to show labels.
Errors	Specify the colors that are used to show script errors.
Line numbers	Specify the colors that are used to show line numbers in the left margin, if line numbering is enabled.
Function call tips	Specify the colors that are used to call tips for InstallScript functions.
Matching braces	Specify the colors that are used to show opening and corresponding closing braces—{ }. This is used whenever the insertion point is within parentheses, brackets, or braces. The syntax coloring takes into account nesting (for example, **Foo(array[array.GetLength() - 1])**).
Unmatched braces	Specify the colors that are used to show an opening brace—{—that is missing its corresponding closing brace—}. This is used whenever the insertion point is after the opening parenthesis, bracket, or brace that is missing a closing mark. The syntax coloring takes into account nesting (for example, **Foo(array[array.GetLength() - 1])**).

To learn more about changing colors in the script editors, see Changing Colors for Syntax Highlighting in the Script Editors.

SCM Pre-shutdown Delay Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The SCM Pre-shutdown Delay dialog box lets you specify the length of the time that the Service Control Manager should wait before proceeding with other shutdown tasks. This dialog box opens in the following scenario:

- You are configuring a service in the Services view, or in the Advanced Settings area for a component in the Setup Design view (installation projects only) or the Components view.
- The **Configure SCM pre-shutdown delay** option is selected in the Configuration Type setting for an event in the Configure Settings category of settings in the right pane, and you click the ellipsis button (...) in the Arguments setting for that event.

Select the appropriate option. To reset the time delay to the default of 3 minutes, select the Default option.

The Service Control Manager waits for the specified period of time after sending the SERVICE_CONTROL_PRESHUTDOWN notification to the service. The change takes effect the next time that the system is started.

Security Descriptor Dialog Box

The Security Descriptor dialog box opens when you click the ellipsis button (...) in the Security Descriptor setting; this setting is displayed under the Permissions setting when you are configuring permissions for a service that you have added to the Services view, or to the Services node in the Advanced Settings area for a component in the Setup Design view (in installation projects) or the Components view.

The Security Descriptor dialog box is where you use valid security descriptor definition language (SDDL) to enter a valid SDDL string for service permissions. For more information about SDDL, see the "Access Control" section of the Microsoft Windows Software Development Kit (SDK).



***Tip** • You can use angle brackets (for example, <DomainName\UserName>) or environment variables (for example, [%UserDomain][%UserName]) to indicate the domain and user name of the user whose account SID is going to be determined at run time.*



***Note** • The permission settings are supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore these settings.*

*An installation should not contain both the **MsiLockPermissionsEx** table and the **LockPermissions** table. If you use the Permissions setting for a service, and the settings under it, you are configuring the **MsiLockPermissionsEx** table of the package. If **Traditional Windows Installer handling** is selected for the Locked-Down Permissions setting in the General Information view and you have set permissions for one or more files, folders, or registry keys in your project, you are configuring the **LockPermissions** table of the package.*

Select File Dialog Box

Use the Select File dialog box to enter a file into the **DuplicateFile** table in the Direct Editor.

Dialog Options

Component

Select the component that contains the file you want to enter in this table.

Files (*.*)

Select the file that you want to include in this table.

Select Media Location Dialog Box

For MSI Databases, choose where you want to place the new file(s) on the source media. If you are creating a .cab file, you can choose to stream the file(s) into the package.



Project *• For MSM Databases, the File Location options are disabled since merge modules require you to store your files in a .cab file that is internal to the .msm database.*



Note *• The option to extract COM information is only enabled when you drag files to a folder or when you add a single file to a component that does not already have any existing files. As a result, the COM file is the only file in the particular component.*

Select SQL Server Dialog Box

The Select SQL Server dialog box opens when you are using the Database Import Wizard and you click the Browse button for the server name on the SQL Server panel.

The Select SQL Server dialog box enables you to select the SQL Server whose database you are importing.

Select String Dialog Box



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The Select String dialog box shows the collection of language-independent identifiers and corresponding language-specific values for the default language of your project. This dialog box lets you select or create a string identifier that you can use in place of a hard-coded text string.

Accessing the Select String Dialog Box

The Select String dialog box is available from within the script editor pane in the InstallScript view. It is also available in other places throughout InstallShield. For example, if you click the ellipsis button (...) in one of the settings that accepts localizable text, the Select String dialog box opens, enabling you to select or create a string entry that you want to use for the selected setting. The Display Name and Description settings for a feature are two examples of a setting that accepts localizable text.

Working with the Select String Dialog Box

The Select String dialog box consists of the following elements:

- A row of buttons and other controls
- A group box area (below the row of buttons)
- A spreadsheetlike table

Each row in the table represents a string entry in your project.

The following table describes all of the buttons and other controls that are displayed in the Select String dialog box.

Table 12-98 • Controls on the Select String Dialog Box

Name of Control	Description
New	Displays the String Entry dialog box, which lets you specify a new string identifier, a value, and internal comments.
Delete	Deletes the selected string entry or string entries.
Search Grid	Filters the strings that are displayed in the Select String dialog box according to the string that you specify in this text box.
Drag a column header here to group that column	Use this group box area to group rows in the Select String dialog box. This dialog box supports multiple levels of grouping simply by dragging the column headings and dropping them onto the group box. InstallShield displays the rows in the dialog box hierarchically according to column arrangement in the group box.

The following table describes each of the columns in the Select String dialog box.

Table 12-99 • Columns in the Select String Dialog Box

Column	Description
Identifier	This column contains the language-independent ID for the string. Each string identifier in a project is linked to one or more values.

Table 12-99 • Columns in the Select String Dialog Box (cont.)

Column	Description
Value	This column shows the run-time string. ***Project*** *• In Basic MSI, DIM, InstallScript MSI, and Merge Module projects, some of the string values contain Windows Installer properties inside square brackets—for example,* ***Install [ProductName]****. At run time, the property and brackets are replaced by the property value.* *String values in these same project types may also contain font information in curly brackets—for example,* ***{&MSSansBold8}OK****. The font information indicates style details that should be used to display the strings at run time.*
Comments	This column contains internal note about the string entries. The comments are not displayed at run time.
Modified	This column lists the date and time that the string entry was last modified.

Serial Number Violation Dialog Box

The Serial Number Violation dialog box opens when InstallShield detects that the serial number entered is already in use. InstallShield will not start for 20 minutes unless a unique, valid serial number is entered.

Task ***To change the serial number:***

1. Click **Change**. The **Serial Number** dialog box opens.
2. In the **Serial Number** box, type a unique, valid serial number. Dashes are not required.

Server Locations Dialog Box

Project *• This information applies to Transform projects.*

The Server Locations dialog box opens when you click the ellipsis button (...) in the Server Locations setting of the General Information view. The Server Locations dialog box lets you specify network and URL paths to the product's installation package and related files so that Windows Installer can find them if they are needed. For example, if an end user installs the product from a network server or Web site and one or more of the features were set to be installed on first use, Windows Installer may need access to the .msi package and related files on the server or Web site sometime after the initial installation.

InstallShield adds the paths that you specify to the **SOURCELIST** property. Windows Installer appends this list to the end of an end user's existing source list for the product at run time.

The validity of the server location that you specify is determined when the installation needs to access the server remotely. That is, if a server is not available, or if you added an invalid server, the entry will be ignored if the resource is needed, and errors might be generated. Each location that is specified must have the complete source for the installation. The entire directory tree at each source location must be the same and must include all of the required source files, including any .cab files. Each location must have an .msi file with the same file name and product code.



***Tip** • Server paths can contain environment variables that are identified with a percent sign (%). For example, the following path uses the Home environment variable:*

\\Server1\%Home%\Office

Table 12-100 • Server Locations Dialog Box Settings

Setting	Description
Source Paths	This box shows the list of paths that you have specified.
New	Click this button to add a new path to the Source Paths box.
Delete	Click this button to delete the selected path from the Source Paths box.
Move Up	Click this button to move the selected path higher in the list of paths.
Move Down	Click this button to move the selected path lower in the list of paths.

Service Options Dialog Box for a Time Delay of an Auto-Start Service



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Service Options dialog box lets you specify whether the Windows service that you are configuring should be started after other auto-start services plus a time delay have elapsed. This dialog box opens in the following scenario:

- You are configuring a service in the Services view, or in the Advanced Settings area for a component in the Setup Design view (installation projects only) or the Components view.

- The **Configure time delay of an auto-start** option is selected in the Configuration Type setting for an event in the Configure Settings category of settings in the right pane, and you click the ellipsis button (...) in the Arguments setting for that event.

Select the appropriate option.

The change takes effect the next time that the system is started.

Service Options Dialog Box for Running Recovery Actions



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Service Options dialog box lets you specify when the recovery actions should be run for the service that you are configuring. This dialog box opens in the following scenario:

- You are configuring a service in the Services view, or in the Advanced Settings area for a component in the Setup Design view (installation projects only) or the Components view.
- The **Configure when to run recovery actions** option is selected in the Configuration Type setting for an event in the Configure Settings category of settings in the right pane, and you click the ellipsis button (...) in the Arguments setting for that event.

Select the appropriate option.

The change takes effect the next time that the system is started.

Service SID Dialog Box



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Service SID dialog box lets you specify the appropriate service security identifier (SID) that you want to add. This dialog box opens in the following scenario:

- You are configuring a service in the Services view, or in the Advanced Settings area for a component in the Setup Design view (installation projects only) or the Components view.
- The **Add a service SID type to the process token** option is selected in the Configuration Type setting for an event in the Configure Settings category of settings in the right pane, and you click the ellipsis button (...) in the Arguments setting for that event.

Select the appropriate option.

Note that if multiple services are hosted in the same process and one service has the restricted service SID, all services must have the restricted service SID.

The change takes effect the next time that the system is started.

Set File(s) URL Dialog Box

The **Set File(s) URL** dialog box opens when you click the **Set File(s) URL** button on the Files to Include tab of the InstallShield Prerequisite Editor. This dialog box lets you specify a file for your InstallShield prerequisite.

Table 12-101 • Set File(s) URL Dialog Box Settings

Setting	Description
URL to file's parent folder	Type the URL for the parent directory of the selected file or files. This URL is the same one that InstallShield uses when installation authors use the Redistributables view to download an InstallShield prerequisite from the Internet to their local machines. For example, if you selected files called Notepad.exe and Paint.exe, and the URLs for these files are http://www.mywebsite.com/Folder1/Notepad.exe and http://www.mywebsite.com/Folder1/Paint.exe, enter the following in this box: http://www.mywebsite.com/Folder1

Settings Dialog Box

The Settings dialog box enables you to set preferences for how InstallShield builds the installation for the project that is currently open.

Task ***To open the Settings dialog box:***

On the **Build** menu, click **Settings**.

The following tabs are associated with this Settings dialog box:

- Compile/Link tab
- Run/Debug tab (InstallScript projects only)

- MSI Log File tab (Windows Installer projects only)

Compile/Link Tab

The Compile/Link tab on the Settings dialog box lets you specify settings for compiling your script. These settings apply to **Setup.rul** and any files included in it.

Table 12-102 • Compile/Link Tab Settings

Setting	Description
Preprocessor Defines	Specify any preprocessor definitions. Use the following format, with no spaces before or after equal signs or commas: **MYVARIABLE1=123,MYVARIABLE2=456** You can test those types of constants in your script by using #if and #ifdef statements that control the flow of the script. For example, type MYVARIABLE in this box, and the code in the following #ifdef loop will be executed: **#ifdef MYVARIABLE** **// Commands** **#endif** After you add or change a preprocessor definition in this box, you must compile your script for the addition or change to take effect. ***Note*** • *Many Windows API functions are declared in the header file* **ISRTWindows.h**, *which is automatically included when you include* **Ifx.h** *in your script. You can prevent the automatic definition of Windows APIs by placing the preprocessor constant ISINCLUDE_NO_WINAPI_H in the Preprocessor Defines box on the Compile/Link tab of the Settings dialog box.*

Table 12-102 • Compile/Link Tab Settings (cont.)

Setting	Description
Include Paths	Specify which directories InstallShield should search for source files that have been included in the main installation's InstallScript code through #include statements. Use the following guidelines for this setting: • To specify multiple paths, separate each one with a comma. • You can use any valid path variable in a path. • Do not use quotation marks to specify a path. • Since the current directory is not necessarily constant, do not use the dot (.) operator in the paths that you specify. Instead, use an explicit path variable such as *<ISProjectFolder>* to indicate a known folder location. • When InstallShield passes the values that you have specified in this setting to **Compile.exe**, InstallShield automatically parses the specified string to determine comma-delimited paths. In addition, InstallShield handles path variable replacement, surrounds the paths with quotation marks, and then passes each path separately to **Compile.exe** using multiple `/i` switches. However, if you separate the paths with semicolons instead of commas, InstallShield considers the specified paths as a single path, and it passes the single path string to the compiler as is. In this case, no path variable substitution or quotation mark additions occurs. ***Note*** • *InstallShield searches any paths that you specify before it searches the standard InstallShield paths. Thus, if two include files share the same name but one include file is in one of your custom include paths and one is in a standard InstallShield include path, InstallShield links to the include file in the custom include path, not the version in the standard InstallShield include path.* For more information, see #include.
Max Warnings	Specifies the maximum number of warning messages that are displayed.
Maximum Errors	Specifies the maximum number of error messages that are displayed.
Warning Level	Specifies the types of warnings that are displayed in the Output window. • **None**—Displays no warning messages. • **Level** 1—Displays any system warning message that InstallShield is unable to handle. • **Level 2**—Displays Level 1 messages, plus a message if string length exceeds the limit. • **Level 3 (Default)**—Displays all warning messages. For more information, see Defining Constants Through the Compiler.

Table 12-102 • Compile/Link Tab Settings (cont.)

Setting	Description
Compile before build	To compile your InstallScript automatically every time that you build a release, select this check box. Note the following behavior that occurs if you select this check box: • If you select the Refresh Build command on the Build menu, your script is recompiled only if changes have been made to it since the last compile. • If you select the Build *<name of release>* command on the Build menu, your script is recompiled, even if no changes have been made to it since the last compile.  ***Note*** • *The compiled script is not copied to the disk images folder before the build begins because the build process automatically copies the compiled script to the folder during the build. If the script fails to compile because of compiler errors, the build does not continue.*
Generate inline debugging information	If this check box is selected, debugging information is included in the compiled script file so a debugging information file is not needed. If you want to debug an installation that includes an object that you have created, and you want to obtain debugging information from the object, you must have compiled the object's script with this check box selected.  ***Caution*** • *Selecting this check box makes the compiled script larger and the installation slower, and it makes it easier for others to reverse engineer your code. Therefore, in most cases, it should not be used when you are creating your final installation for distribution to end users.*
Warnings as errors	If warnings should be treated as errors (that is, if the installation should not be run if there were any compile warnings when the InstallScript was compiled), select this check box.

Table 12-102 • Compile/Link Tab Settings (cont.)

Setting	Description
Additional Library Paths	 ***Project*** *• For InstallScript and InstallScript Object projects, InstallShield automatically searches for script libraries in the following directories at compile time:* • **<ISProductFolder>\Script\Ifx\Lib** • **<ISProductFolder>\Script\Isrt\Lib** *For Basic MSI and InstallScript MSI projects, InstallShield automatically searches for script libraries in the following directories at compile time:* • **<ISProductFolder>\Script\Iswi\Lib** • **<ISProductFolder>\Script\Isrt\Lib** *The aforementioned directories contain the libraries that define the built-in InstallScript functions and that are listed in the Libraries (.obl) box by default.* If you want InstallShield to search any additional directories that contain your own script libraries, you can specify the directory path in the Additional Library Paths box. Use the following guidelines for this setting: • To specify multiple paths, separate each one with a comma. • You can use any valid path variable in a path. • Do not use quotation marks to specify a path. • Since the current directory is not necessarily constant, do not use the dot (.) operator in the paths that you specify. Instead, use an explicit path variable such as *<ISProjectFolder>* to indicate a known folder location. • When InstallShield passes the values that you have specified in this setting to **Compile.exe**, InstallShield automatically parses the specified string to determine comma-delimited paths. In addition, InstallShield handles path variable replacement, surrounds the paths with quotation marks, and then passes each path separately to **Compile.exe** using multiple `/libpath` switches. However, if you separate the paths with semicolons instead of commas, InstallShield considers the specified paths as a single path, and it passes the single path string to the compiler as is. In this case, no path variable substitution or quotation mark additions occurs.  ***Note*** *• InstallShield searches any paths that you specify before it searches the standard InstallShield paths. Thus, if two libraries share the same name but one library is in one of your custom library paths and one is in a standard InstallShield library path, InstallShield links to the library in the custom library path, not the version in the standard InstallShield library path.*

Table 12-102 • Compile/Link Tab Settings (cont.)

Setting	Description
Libraries (.obl)	Specify the names of script libraries (compiled .obl files) to which you want the compiler to link. Separate each file with a comma. ***Note*** • *It is not necessary to include the path of each library, as long as either the path is identified explicitly in the Additional Library Paths box or the path is one of the built-in library paths that InstallShield automatically searches at compile time.* ***Project*** • *The following built-in libraries are listed by default for InstallScript projects: ifx.obl,isrt.obl.* *The following built-in libraries are listed by default for Basic MSI and InstallScript MSI projects: iswi.obl,isrt.obl.* *The following built-in libraries are listed by default for InstallScript Object projects: ifxobject.obl,isrt.obl.* *Additional built-in libraries may also be listed, depending on which views in InstallShield you use when you are developing your project.*

Run/Debug Tab

Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The Run/Debug tab lets you configure settings relevant to the running and debugging of your installation.

Table 12-103 • Run/Debug Tab Settings

Settings	Description
Setup Command Line Arguments	Text placed in this box is written to the system variable CMDLINE when you run the installation from within InstallShield; this variable can be accessed in the script.
Generate MIF File	If you select this check box, the installation automatically generates a Management Information Format (.mif) file when you run the installation from within InstallShield. For information on generating .mif files in the installation that you distribute, see The [Mif] Section.
MIF Filename	Specifies the name of the .mif file generated by the installation when you run the installation from within InstallShield. If you leave this box blank, the .mif file is named Status.mif.

Table 12-103 • Run/Debug Tab Settings (cont.)

Settings	Description
Product Serial Number	If you specify a serial number, the installation adds it to the .mif file it generates when you run the installation from within InstallShield.

MSI Log File Tab

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The MSI Log File tab on the Settings dialog box enables you to create a log file that you can use when you are running your installation from within InstallShield. In addition, this tab lets you select the type of information that you would like written to the log file.

Table 12-104 • MSI Log File Tab Settings

Setting	Description
MSI Log File Options	Select the check boxes for the types of messages that you want to be displayed in your log file. When you select a check box, the corresponding argument is added to the MsiExec.EXE Command-Line Arguments box.
MsiExec.EXE Command-Line Arguments	Enter the command-line arguments that you would like to pass to **MsiExec.exe** when your installation is run. When check boxes are selected in the MSI Log File Options area, the corresponding command-line parameters are added to this box.
Log File	Type the name of the log file that will be created when your installation is run. ***Project*** • *For InstallScript MSI projects, you need to type the full path to the log file in this box.*

Setup Languages Dialog Box

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

- *Merge Module*

The behavior of the Setup Languages dialog box varies, depending on whether the project type is Windows Installer based (a Basic MSI, InstallScript MSI, or Merge Module project) or InstallScript based (an InstallScript or InstallScript Object project).

The Setup Languages dialog box opens when you click the ellipsis button (...) for the Setup Languages setting in the General Information view.

In Basic MSI, InstallScript MSI, and Merge Module projects, the Setup Languages dialog box lets you specify the languages that you want to be listed in the UI Languages setting in the Releases view. Therefore, if a language is not listed in this dialog box at the project level, you cannot include that particular UI language in your project's releases.

In InstallScript and InstallScript Object projects, the Setup Languages dialog box lets you specify the languages that you want to be listed in the Languages setting in the Components and Releases views in your project. In general, if a language is not listed for this setting at the project level, you cannot designate that a particular component in your project is targeted for that language; in addition, you cannot designate that the components and UI strings for a particular language are included in your project's releases.

When you add a supported language to your project through this setting, InstallShield adds string entries for that language to your project. The string entries include the built-in user-interface string resources that are already translated.

Table 12-105 • Settings on the Setup Languages Dialog Box

Setting	Description
Languages	If you want the strings of a particular language to be included in your project, select its check box. To exclude all of the strings for a particular language, clear its check box. ***Edition*** • *Language support varies, depending on the edition of InstallShield that you are using, the project type, and the language of the InstallShield interface (English or Japanese). For example, if you are using the English version of InstallShield Professional Edition, only one language is available for selection in this dialog box; if other language check boxes are listed, they are disabled.* *For general information about language support, see Run-Time Language Support in InstallShield.*
Show only selected languages	If you want the languages box to list only languages whose check boxes are selected, select this check box.
Show only available languages	If you want the languages box to list only languages that are available to you with the edition of InstallShield that you are using, select this check box.

SQL Server Requirement Dialog Box

Use the SQL Server Requirement dialog box to define the server requirements for your database.

Predefined Minimum Database Server Version

Select a predefined minimum database server version that your application supports. Choose from the list of available options.

Note • *At runtime, the installation verifies that all database servers that are selected by the end user meet the specified requirements.*

Customize

Rarely, you will want to define your own database server version, but in the cases that you must, check the Customize box and any of the following options that apply:

Table 12-106 • Customize Options

Option	Description
Major Version	This is the major version returned by a SQL Server. For example, SQL Server version 7 returns 7.00
Service Pack Level	This is the server pack level number returned by a SQL Server. For example, SQL Server version 7 RTM returns 623.
Any Greater than Major Version	Check this option to support future major versions.
Any Greater than Service Pack Level	Check this option to support future service packs.

Supersedence Dialog Box

Edition • *This information applies to Basic MSI projects.*

The Supersedence dialog box opens when you click the ellipsis button (...) in the Supersedence setting of the General Information view. This dialog box lets you specify one or more other .msi packages that you want to be superseded by the installation that you are creating. InstallShield includes the list of .msi packages that you provide in a software identification tag, which provides data that AdminStudio includes when users import packages into the AdminStudio Application Catalog. AdminStudio makes this information available to users for review and tests before publishing to Microsoft System Center 2012 Configuration Manager.

When you are selecting superseded .msi packages, you can specify earlier versions of the current installation. You can also optionally select .msi packages of other tools. For example, if you are working on an installation for an image editing tool, you can have your installation supersede other image editing tools that your enterprise no longer supports.

To specify an .msi package that should be superseded by the current installation, click the Add Superseded .msi File button, and then select the package that you want to include in the list.

UI Languages Dialog Box

Edition • *This information applies to the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*
- *Suite/Advanced UI*

The UI Languages dialog box lets you specify which user interface languages you want to include in the release that is selected in the Releases view. To open this dialog box, open the Releases view and select a release. Then click the ellipsis button (...) in the UI Languages setting on the Build tab.

Note that if a language is not selected in the Setup Languages setting in the General Information view of the project, it is not listed as one of the available languages for the UI Languages setting.

Update Merge Module Search Path Dialog Box

The Update Merge Module Search Path dialog box opens when you use the Browse for Merge Module option to add a merge module to your installation project. It displays the path that will be added to the merge module search path.

Click OK to approve the action.

To learn more about why this dialog box appears, see What Happens When You Browse for a Merge Module.

Wizard Reference

InstallShield includes many wizards to assist you in creating your installation project.



Project • *Not all wizards are available with all project types.*

The following wizards are available in InstallShield.

- Component Wizard
- Convert Source Paths Wizard
- Create New QuickPatch Wizard
- Create Table Wizard
- Custom Action Wizard
- Database Import Wizard
- Device Driver Wizard
- Dialog Wizard
- DirectX Object Wizard
- Dynamic Scanning Wizard
- Export Components Wizard
- Function Wizard
- Import DIM Wizard
- Import REG File Wizard
- Import XML Settings Wizard
- Microsoft .NET Framework Object Wizard
- New Language Wizard
- Open MSI/MSM Wizard
- Open MSP Wizard
- Open Transform Wizard
- Publish Wizard
- Redistributable Downloader Wizard
- Reg-Free COM Wizard
- Release Wizard
- Setup Best Practices Wizard
- Static Scanning Wizard
- System Search Wizard

- Transform Wizard
- Upgrade Validation Wizard
- User Interface Wizard
- Visual Studio .NET Wizard for Visual Basic .NET, C# .NET, and Visual C++ .NET
- Visual Studio Deployment Project Import Wizard

Component Wizard



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Although there are several ways to create Windows Installer components and specify advanced settings for them, the Component Wizard simplifies the process. The wizard assists you by organizing your files into components and making settings for files with advanced installation requirements—often by automatically retrieving information about the files.



Task

To launch the Component Wizard, do one of the following:

- For installation projects only: In the **Setup Design** view, right-click a feature and click **Component Wizard**.
- For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, and MSM Database projects: In the **Components** view, right-click the **Components** explorer and click **Component Wizard**.

When you launch the Component Wizard, you are presented with the following options on the Welcome panel:

Table 12-1 • Welcome Panel Settings

Setting	Description
Create components for me using Best Practices	Select this option to specify all of your application's files and have InstallShield create all of the necessary components for you according to Setup Best Practices.

Table 12-1 • Welcome Panel Settings (cont.)

Setting	Description
Let me select a type and define the component myself	This option creates components containing files that need special treatment upon installation and uninstallation. The wizard can create components for only one of the following file categories at a time. Run the Component Wizard again to build components for another type of file. • COM Server • Install Service • Control Service • Fonts  ***Note*** • *If you want to create a component that installs, starts, stops, or deletes a service during installation or uninstallation, you can use the Services view. You can also use this view to configure extended service customization options, which are not configurable through the Component Wizard. For more information, see Installing, Controlling, and Configuring Windows Services.*

Welcome Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Although there are several ways to create Windows Installer components and specify advanced settings for them, the Component Wizard simplifies the process. The wizard assists you by organizing your files into components and making settings for files with advanced installation requirements—often by automatically retrieving information about the files.

Tip • *The Component Wizard should not be used to modify existing components. When you have created your components, you can edit their properties, files, and advanced settings in the Setup Design view or the Components view.*

Table 12-2 • Welcome Panel Settings

Setting	Description
Create components for me using Best Practices	Select this option to specify all of your application's files and have InstallShield create all of the necessary components for you according to Setup Best Practices.
Let me select a type and define the component myself	This option creates components containing files that need special treatment upon installation and uninstallation. The wizard can create components for only one of the following file categories at a time. Run the Component Wizard again to build components for another type of file. • COM server • Install Service • Control Service • Fonts ***Note*** • *If you want to create a component that installs, starts, stops, or deletes a service during installation or uninstallation, you can use the Services view. You can also use this view to configure extended service customization options, which are not configurable through the Component Wizard. For more information, see Installing, Controlling, and Configuring Windows Services.*

Setup Best Practices

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

This Component Wizard path is the easiest means for organizing all of your application's files into components. The wizard also handles installing files with special installation requirements, such as COM servers and fonts, if it detects them among the files that you provide. For more information on how the Component Wizard organizes your files according to Best Practices, see Best Practice Rules for Creating Components.

After selecting the Best Practices option in the Component Wizard, the next panel you see is the Best Practices–Destination Panel.

Destination Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

The first step is to specify the folder where the components' files will be installed on the target system.



Note • *The Destination setting is disabled if you launch the Component Wizard by right-clicking on a destination in the Files and Folders view. The destination that you selected before launching the Component Wizard becomes the destination for the component that you create.*

Table 12-3 • Destination Panel Settings

Setting	Description
Destination	The wizard uses the same destination folder for all of the components it creates. The destination you select in this field becomes the Destination Folder setting for every component that is created in this wizard session. Select a Windows Installer folder property, in square brackets, from the drop-down list of standard destination folders. The default value points to the root of INSTALLDIR, which is initialized in the product's Destination Folder setting. Select **Browse, create, or modify a directory entry** from the drop-down list to display the Browse for Directory dialog box.

Files Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

- *MSM Database*

In the Files panel, specify the files that should be grouped into components.

Table 12-4 • Files Panel Settings

Setting	Description
Files	The Files list displays information about each file associated with this component. Add files to the list in either of the following ways: • Drag and drop files from Windows Explorer onto the file list. • Click Add Files and browse to your application's files. ***Note*** • *The path to each file is a hard-coded link. If the file is moved or renamed, the Link To value reads *** File Not Found ***, and the Release Wizard will be unable to resolve the file link when you build your setup package.*
Extract COM Info Immediately	If this option is selected, the component's COM Extract at Build setting is set to No and the COM data is extracted immediately. If you clear this check box, the COM data is extracted at build time.
Add Files	Click Add Files to browse to the files you want to add to the list. In the resulting dialog you can select as many files as you want from a single folder by pressing the SHIFT or CTRL key and clicking on the files. You can also use the Insert key to add files.
Add Folder	Click Add Folder to browse to a folder that you would like to add to the list. All the files under this folder, including any files in subfolders, will be added to your project.
Remove Files	Select one or more files and click Remove Files to permanently delete them from the list. You can also use the Delete key to delete files.

Summary Panel

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

- *MSM Database*

The Summary panel displays the components' settings. Review the summary, and click Finish to complete the component creation process and organize the files into components.

If you launched the wizard by right-clicking a feature, the new components are associated with that feature. Otherwise, you must open the Setup Design view, right-click one or more features, and select Insert Components to associate your new components with features.

The Component Wizard provides smart defaults for most of your component settings and any necessary advanced settings. To edit them, select a component in the Components view and modify the component's settings.



Note • *Components that are not associated with a feature are displayed with the orphaned component icon.*

Component Type Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

The Component Type panel enables you to create components of a single type. Since these components all have unique installation and uninstallation requirements, subsequent panels of the Component Wizard differ, depending on the type of component that you are installing.

Table 12-5 • Component Type Panel Settings

Setting	Description
Component Name	When you click a component type, the wizard suggests a unique name for your new component in the Component Name field. To change the name, type a new name over the suggested name.  ***Note*** • *Although you can specify a single component type for multiple font files or multiple services in one file, these are all placed into a single component and require only one component name.* *This setting is disabled if you are creating the font type of component. InstallShield uses the font file name as the name of font components.*

Table 12-5 • Component Type Panel Settings (cont.)

Setting	Description
Component Types	Select the type of component that you want to create. • COM Server • Install Service • Control Service • Fonts ***Note*** • *If you want to create a component that installs, starts, stops, or deletes a service during installation or uninstallation, you can use the Services view. You can also use this view to configure extended service customization options, which are not configurable through the Component Wizard. For more information, see Installing, Controlling, and Configuring Windows Services.*

COM Server Component Type

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

The COM server component type in the Component Wizard accepts a single COM server (.exe, .dll, or .ocx) file and creates a component for it in your installation project.

Since these files require special registration so that they can operate on the target system, the wizard attempts to extract the information for you. If the wizard fails to extract the file's registry settings, it prompts you for the CLSID, progID, and so on.

All of the registration information that maps to the COM Registration advanced setting is written there. The wizard creates any additional entries, such as the InProcServer32 ThreadingModel value, in the component's registry data.

Tip • *If you are using InstallShield on a 64-bit system, InstallShield can extract COM data from a 64-bit COM server. In order to install the data to the correct locations, the component must be marked as 64 bit. To learn more about 64-bit support, see Targeting 64-Bit Operating Systems.*

Using the component's data, the installer registers the file during installation and unregisters it during uninstallation instead of calling self-registration functions, which would be a violation of Setup Best Practices.

***Note** • The PATH system variable on the build machine must be set to include the directories of all of the .dll files to which the COM server links; otherwise, the file will fail to register and COM information will not be extracted.*

The wizard asks if your COM server runs as a service, so you can specify installation parameters for the service.

COM Server File Panel

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

The COM Server File panel is where you identify basic information about your COM server's executable file.

Table 12-6 • COM Server File Panel Settings

Setting	Description
COM Server File	Enter the path and file name of your COM server (.exe, .dll, or .ocx) file, or click the Browse button to navigate to the file. You can specify only one portable executable file per component according to Setup Best Practices. Launch the wizard again to install additional COM servers, or create the components using the Component Wizard's Best Practices option.

Table 12-6 • COM Server File Panel Settings (cont.)

Setting	Description
Extract registration information	Select this check box to have the wizard scan your file to extract all of the file's registration information. If the extraction is successful, the next panel displayed is the Summary confirmation dialog. ***Project*** • *If you are working on a Basic MSI, InstallScript MSI, or Merge Module project, an alternative to extracting the COM information through the wizard and maintaining it in the COM Registration advanced setting is to have the data dynamically extracted at build time. If you are working in Direct Edit Mode, COM information cannot be extracted at build time; it is extracted at design time.* If you do not select this check box or if the wizard cannot read the file's registration, the Classes panel is displayed next. ***Tip*** • *If you are using InstallShield on a 64-bit system, InstallShield can extract COM data from a 64-bit COM server. In order to install the data to the correct locations, the component must be marked as 64 bit. To learn more about 64-bit support, see Targeting 64-Bit Operating Systems.*
This file also runs as a service	This check box is enabled only if your COM server is an .exe file. If you select this option, the wizard displays panels that enable you to specify installation parameters for services. If you select this check box and specify the service information, InstallShield enters the information in the Services area under the Advanced Settings node of your component in the Components view. You do not have to launch the wizard again to create a service component for this COM server.
Custom EXE Command Line	This check box is enabled only if your COM server is an .exe file. Once it is enabled, you can enter a command line argument. The command line argument overrides the default `/regserver` command line to an .exe file. If you select the service option and specify parameters in the Custom EXE Command Line box, `/service` is passed as the command line. By default, the `/service` command line does everything that the `/regserver` command line does but it also adds registry values for service support. If you select all the options on this panel, the command line that you enter in this box is the command line that is passed to the .exe file.

If the COM server is part of a 64-bit component and you are running InstallShield on a 64-bit machine, InstallShield performs 64-bit COM extraction. For more information, see Targeting 64-Bit Operating Systems.

Classes Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

This panel is where you begin providing registration information for your COM server—including your file's classes, identified by their progIDs and CLSIDs.

This panel and the next few registration information panels are displayed only if you elected not to have the wizard extract the information, or if the wizard failed for any reason (for example, if the .exe, .dll, or .ocx file does not indeed contain a COM server). Otherwise, the wizard would continue with the Service Executable panel—if the COM server is a service, or the Summary panel—if the COM server is not a service.

Table 12-7 • Classes Panel Settings

Setting	Description
ProgID	Enter the COM server's programmatic identifier, or progID in this panel. Each progID corresponds to a COM class in your file. To add a new progID in the format Program.Component.N, click the Add button or press the Insert key. Enter the same value in both the ProgID and Version-Independent ProgID fields if the progID is version independent (in the format Program.Component). All of the other information in this panel corresponds to the class you named in this field with a progID. Click on each progID to set its class ID and version-independent progID and to check whether it is DCOM/COM+ enabled.
Add	Click Add to register a new class for this COM server. To rename the service, type the new name immediately after clicking the Add button, or press F2 and then type the new name. You can also use the Insert key to add a new service.
Remove	Click Remove to permanently delete a progID from the list. You can also use the Delete key to remove a service.
Class ID	Click on a progID to provide the corresponding class ID. Enter a string GUID for the CLSID that Windows Installer will register for this class.

Table 12-7 • Classes Panel Settings (cont.)

Setting	Description
Version-Independent ProgID	Click on a progID to provide the corresponding version-independent progID, if any, in the format *Program.Component*. If the progID itself is version independent, enter the same progID in this field.
This COM server is DCOM/COM+ enabled	If the COM class is enabled for DCOM or COM+, and therefore requires additional registry entries, check this box. The wizard prompts you for this information in the AppID General Information and AppID Advanced Information panels.

Context Types Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Indicate the type of file in which your COM server resides. The available context types are:

- InprocServer—16-bit DLL or OCX
- InprocServer32—32-bit DLL or OCX
- LocalServer—16-bit EXE
- LocalServer32—32-bit EXE

Type Library Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Specify the type library, if any, referenced by this COM server.

Table 12-8 • Type Library Panel Settings

Setting	Description
Type Library Description	Provide a brief description of the type library. This value is used in the component's COM Registration advanced setting as the type library's name in the COM Registration Explorer. The description is registered as the default value under HKEY_CLASSES_ROOT\TypeLib*libID**version*.
Type Library GUID	Enter the GUID (LibID) that identifies this type library. You can paste (Ctrl+V) your COM server's LibID into this field.
Language	Enter the decimal value of the type library's locale ID. Since type libraries are usually language neutral, 0 (zero) is the most common language of type libraries.
Version	Provide the version number of the type library. If this field is blank, Windows Installer automatically determines the version of the type library at run time.

AppID General Information Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

In this panel, you provide basic information about your DCOM or COM+ file's AppID. An AppID is a GUID registered under HKEY_CLASSES_ROOT\AppID used for grouping all of the security and configuration options of distributed COM objects. The information you provide in this panel and in the AppID Advanced Information panel are registered under this key.

This panel is not displayed unless you selected the **This COM server is DCOM/COM+ enabled** check box on the Classes panel.

Table 12-9 • AppID Information Panel Settings

Setting	Description
AppID GUID	Enter the string GUID (AppID) for this COM+ or DCOM component. You can paste (Ctrl+V) the AppID into this field.

Table 12-9 • AppID Information Panel Settings (cont.)

Setting	Description
Remote Server Name	Enter the name of the computer on which this component should run.

AppID Advanced Information Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

In this panel, you provide more advanced information about your DCOM or COM+ file's AppID than you provided in the AppID General Information panel.

This panel is not displayed unless you selected the **This COM server is DCOM/COM+ enabled** check box on the Classes panel.

Table 12-10 • AppID Advanced Information Panel Settings

Setting	Description
Activate at Storage	Select this check box to indicate that the distributed COM server resides on the same machine as its data. When a DCOM file is activated at storage, its objects are instantiated on the same machine as the data it uses, thereby reducing network traffic. Note that calling applications can override this default behavior with DCOM API functions.
Run as Interactive User	Select this check box to have the COM server run under the same account as the user currently logged onto the system. This option allows the server to be connected with the interactive desktop. Since COM servers that also run as services are not designed to run as the interactive user, this check box is disabled if you select the Run as a Win32 Service check box.

Table 12-10 • AppID Advanced Information Panel Settings (cont.)

Setting	Description
Run as a Win32 Service	Select this option if the distributed COM .exe file is intended to run as a service. When this option is selected, you should provide the name of the service and any parameters it takes. ***Note*** • *If you indicated—in the COM Server Executable panel—that this file runs as a service, the next panel displayed is the Service Executable panel, in which you must enter information about this file's services.*
Service Name	Enter the name of the service.
Service Parameters	Enter the parameters you want passed to the service when it is launched.

Summary Panel

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Review the component's description and click Finish to have the wizard create the described components. Click Back if you need to change any of the information you provided.

All of the registration information that maps to the COM Registration advanced setting is written there. The wizard creates any additional entries, such as the InProcServer32 ThreadingModel value, in the component's registry data.

If the wizard lists **Please enter this information in the IDE**, go to the component's COM Registration (or Install Services for components that also run as a service) advanced setting and provide the values.

If you launched the wizard by right-clicking a feature, the new components are associated with that feature. Otherwise, you must open the Setup Design view, right-click one or more features, and select Insert Components to associate your new components with features.

Note • *Components that are not associated with a feature are displayed with the orphaned component icon.*

Control Service Component Type

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Note • *If you want to create a component that installs, starts, stops, or deletes a service during installation or uninstallation, you can use the Services view. You can also use this view to configure extended service customization options, which are not configurable through the Component Wizard. For more information, see Installing, Controlling, and Configuring Windows Services.*

The Control Service component type is used start or stop a single service during installation or uninstallation. For example, you might need to delete a running service before updating it in your setup.

The wizard takes you through a series of panels prompting you for information about the service you want to control. Once you have completed the wizard, you have a new component consisting only of default properties and the advanced settings necessary for controlling a service. Later you can add data such as files, shortcuts, and registry entries to your component through the Setup Design or Components view.

Task ***To control another service:***

Launch the wizard again or modify the component's Control Service advanced setting.

Task ***To install a service:***

Launch the wizard again and select the Install Services component type or add the service's file to the Control Service component you have created in the wizard and edit its Services node in the Advanced Setting area of the Components view.

Specify Service Panel

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

The first step is to specify which service you want to install. Everything else you do in the wizard refers to this service.

You can use the Component Wizard to control a single service at installation and uninstallation. To control another service in a new component, launch this wizard again.

Note • *To control more than one service in the same component, you must modify the component's Services node under the Advanced Settings area of the Components view.*

Table 12-11 • Specify Service Panel Settings

Setting	Description
Service is present on target system	Select this option to control a service that you are certain will be installed on the target system when your setup runs or that will be when your product is uninstalled.
Service Name	Enter the name of the service. This is the name that InstallShield, Windows Installer, and the Service Control Manager (which maintains the system's database of services and exposes an interface for controlling these services) use to identify a service—not its display name.
Service is included in this setup	Select this option to choose from among the services you have created in your setup project. ***Note*** • *This option is disabled if none of your components has a service in its Services node under the Advanced Settings area in the Components view.*

Installation Events Panel

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

This panel offers you standard options for controlling the service during your product's installation.

Table 12-12 • Installation Events Panel Settings

Setting	Description
Trigger the following events when the component is installed	Click this option to enable the installation control events.

Table 12-12 • Installation Events Panel Settings (cont.)

Setting	Description
Start the service	This event starts the service when the component is installed. Specifically, it is started by the Start Services action in your setup's Installation sequence. Specify any command-line parameters to be used for launching the service in the Arguments field.
Arguments	Specify any arguments that you want passed to the service when it is started. Separate each argument with a comma.
Stop the service	This event stops the service when the component is installed. It is stopped by the Stop Services action in your setup's Installation sequence.
Delete the service	This event deletes the service when the component is installed. It is deleted by the Delete Services action in your setup's Installation sequence.
No event (do nothing)	Select this option if you do not need to control this service during your product's setup. This option cancels any of the above events.

Uninstallation Events Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

This panel is prompting you for the same control events that were available for the product's installation, except that they will be triggered only when this component is uninstalled.

Table 12-13 • Uninstallation Events Panel Settings

Setting	Description
Trigger the following events when the component is uninstalled	Click this option to enable the uninstallation control events.
Start the service	This event starts the service when the component is uninstalled. Specifically, it is started by the Start Services action in your setup's Installation sequence. Specify any command-line parameters to be used for launching the service in the Arguments field.

Table 12-13 • Uninstallation Events Panel Settings (cont.)

Setting	Description
Arguments	Specify any arguments that you want passed to the service when it is started. Separate each argument with a comma.
Stop the service	This event stops the service when the component is uninstalled. It is stopped by the Stop Services action in your setup's Installation sequence.
Delete the service	This event deletes the service when the component is uninstalled. It is deleted by the Delete Services action in your setup's Installation sequence.
No event (do nothing)	Select this option if you do not need to control this service during your product's setup. This option cancels any of the above events. If you selected the **No event (do nothing)** option on the Installation Events panel, it is disabled in this panel. You must select a control event at either installation or uninstallation.

Wait Type Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

In this panel, you can set the wait option for all events in your product's installation and uninstallation. Windows Installer proceeds as specified in this panel, following the control event.

Select **Wait for the service to complete the event before continuing** if the event is critical to the installation or uninstallation.

If you selected the **No event (do nothing)** option in the Installation Events or Uninstallation Events panel, the corresponding wait settings are disabled in this panel.

Summary Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

- *MSI Database*
- *MSM Database*

Review the information the wizard has collected, and click Finish to create your Control Service component.

You can review and edit the new component's settings in the Components view or the Setup Design view.

If you launched the wizard by right-clicking a feature, the new components are associated with that feature. Otherwise, you must open the Setup Design view, right-click one or more features, and select Insert Components to associate your new components with features.



***Note** • Components that are not associated with a feature are displayed with the orphaned component icon.*

Install Service Component Type



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*



***Note** • If you want to create a component that installs, starts, stops, or deletes a service during installation or uninstallation, you can use the Services view. You can also use this view to configure extended service customization options, which are not configurable through the Component Wizard. For more information, see Installing, Controlling, and Configuring Windows Services.*

The Install Service component type is used to create a component that contains the information necessary to install all of the Win32 services that are present in a single .exe file.



***Tip** • If the file also runs as a COM server, a better way to create the component is to select the COM Server component type and specify that it contains a service. Once you have provided all the COM registration data, the Component Wizard will the prompt you for the same information about the service that you must provide for the Install Services component type.*

The wizard takes you through a series of panels prompting you for information about the services you want to install. Once you have completed the wizard, you have a new component consisting of your file, the default component properties, and the advanced settings necessary for installing and uninstalling the services.

You can edit the services' installation information in the Services node under the Advanced Setting area of the Components view. Windows Installer will pass these values to the Windows API function CreateService() to install each service onto the target system and register it with the Service Control Manager (which maintains the system's database of services and exposes an interface for controlling these services).

Service Executable Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Provide the fully qualified path for your file and list all of its services. Only .exe files are allowed. Driver services (.sys files) are not supported in the wizard.

Table 12-14 • Service Executable Panel Settings

Setting	Description
Service Executable	Enter the fully qualified path to the executable file for the services in this component or click the Browse button to navigate to the service's executable file. In compliance with Setup Best Practices, the component can contain only one executable file. Launch the wizard again later to add another file to a new component. If you have arrived at this wizard panel as the result of having a COM Server component that contains services, the Service Executable box is disabled. The wizard uses the same executable that you entered in the COM Server Executable panel.  ***Caution*** • *If you use the Browse button to navigate to and select a file, the new value overwrites anything that was entered in the Service Executable box.*
Services	This list contains all of the services found in the file.
Add	Click Add to add a new service. To rename the service, type the new name immediately after clicking the Add button, or press F2 and then type the new name. This is the name that InstallShield, Windows Installer, CreateService(), and the Service Control Manager (which maintains the system's database of services and exposes an interface for controlling these services) use to identify the service—not its display name. You can also use the Insert key to add a new service.
Remove	Select a service and click Remove to permanently delete it from the Services list. You can also use the Delete key to remove a service.

Service Type Information Panel

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Select each service in the drop-down list and provide the necessary details.

Table 12-15 • Service Type Information Panel Settings

Setting	Description
Service	Select the services that you entered in the Services list of the Service Executable panel and then specify the Display Name and Service Type installation properties for each.
Display Name	Enter a display name for the service. This is the name that is displayed to users in the SCM. If you leave the Display Name field blank, the name you gave the service is used as the display name. ***Note*** • *Other service properties, such as the service description to display on target systems, are available in the component's advanced settings.*
Service Type	Specify whether the service runs in its own process or shares a process with others. Select the **Service shares a process with others** option only if there is more than one service in the component. If you are familiar with the source code for the service, select **This service runs in its own process** if it is of type SERVICE_WIN32_OWN_PROCESS, and **Service shares a process with others** if it is of type SERVICE_WIN32_SHARE_PROCESS. ***Note*** • *You are limited to the options provided under Service Type. The Component Wizard does not support driver services.*
Arguments	Enter command-line arguments to pass to the service when it runs. Expressions of the form [PropertyName] will be expanded to the value of the Windows Installer property called PropertyName.

Service Start Type Information Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Select each service in the drop-down list and indicate the way in which you want the service started—or if you want the service disabled.

Start Type

Select the service's start type from the list.

- Automatically when the system starts up
- On demand through the Service Control Manager
- Not started (Disabled)

Service Load Order Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Select each service in this file and enter its load-ordering group and dependency information. Having dependencies and membership in a load-ordering group means that the service requires other services to be running before it can be started. (Service load-ordering groups are listed under HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\ServiceGroupOrder. The service's Start Type property determines when it loads within its group.)

Table 12-16 • Service Load Order Panel Settings

Setting	Description
Service	Select each of the services that you entered in the Services list of the Service Executable panel and then specify its load-ordering and dependency information.

Table 12-16 • Service Load Order Panel Settings (cont.)

Setting	Description
Load-Ordering Group	Enter the name of the load-ordering group of which this service is a member. Leave this field blank if the service does not belong to a group.
Dependencies	List the service's dependencies in the edit field. A service's dependency can be another service—in which case enter the service's name, or it can be a load-ordering group—in which case precede the name of the load-ordering group with a plus sign (+) so that the SCM can distinguish it from a service. Separate each dependency with a comma.

Error Control Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Select each of the services that you entered in the Services list of the Service Executable panel and—for each service—specify which action you would like the Service Control Manager to take if it receives an error while attempting to start the service.

- Log the error and continue starting the service.
- Log the error, display a message box, and continue starting the service.
- Log the error and, if possible, restart the system with the last known good configuration.

Service Logon Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Select each service in the list and specify the type of account you want the service to log on as when it starts up.



Caution • *Because of potential password conflicts, Microsoft recommends that all services log on under the local system account.*

Table 12-17 • Service Logon Panel Settings

Setting	Description
Service	Select each of the services that you entered in the Services list of the Service Executable panel and then specify how the service should log on.
Local system account	Select this option to have the service log on under the local system account.
Allow service to interact with the desktop	If the service has a user interface, click this option.
This account	Select this option if you want the service to impersonate a specific user when it logs on. You must enter a user name and password, or the local system account will be used.
Domain/User Name	Enter accounts in the form **DomainName\UserName**. You can use a dot for the built-in domain, such as **.\UserName**.
Password	Enter the password for this account. The password will not be used if you do not specify a user name, and the service will log on under the local system account.

Summary Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Review the information the wizard has collected, and click Finish to create your service component.

The wizard provides smart defaults for all of your component's properties. You can review the property sheet in the Setup Design views. Of particular importance for an service is the Remote Installation setting, since Windows Installer cannot install a service that does not reside on the local system.

If you launched the wizard by right-clicking on a feature, the new component is associated with that feature. Otherwise, you must open the Setup Design view, right-click on one or more features, and select Insert Components to associate your new component with a feature.



***Note** • Components that are not associated with a feature are displayed with the orphaned component icon.*

After the wizard finishes, you can modify any of the installation options for the component's Services node under the Advanced Setting area of the Components view.

InstallShield provides functionality for controlling this service or others on the target system. For more information, see Installing, Controlling, and Configuring Windows Services.

Font Component Type



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

Although you can create a component in one of several ways, the easiest way to install a font (a .ttf, .fon, or .ttc file) in a setup is to create a Fonts component in the Component Wizard.

Unlike the other component types offered in the wizard, there is no corresponding advanced setting for Fonts components. The setting that distinguishes a Fonts component and tells Windows Installer to install it differently is the file's Font Title property.

An alternative to creating a Fonts component is to select the Setup Best Practices option in the Component Wizard and let InstallShield create a Fonts component for you from your font files.



***Note** • If you want a font to remain on the system after the application has been uninstalled, set its component's Permanent property to Yes.*

Add Installed Fonts Panel



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

The wizard searches for all fonts that are registered on the development system. You can create a component containing any of these fonts by selecting them in the list.

Table 12-18 • Add Installed Fonts Panel Settings

Setting	Description
Fonts	Select the fonts you want to include in your Fonts component.
I also want to add fonts not installed on this system	Click this option to be able to browse to additional fonts in the Add New Fonts panel. If you do not select this option, the wizard displays the Summary panel next.

Add New Fonts Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

This panel lets you add any .ttf, .fon, or .ttc file to your Fonts component.



Note • *This panel is designed to allow you to add fonts to your installation that are not installed on your machine. For example, you might have a font file that is stored on a network drive. As a result, you cannot select fonts from your Fonts folder since those fonts are already installed on your local machine. If you want to add a font that is installed on your machine to your installation, click Back and select the appropriate font from the list.*

Table 12-19 • New Fonts Panel Settings

Setting	Description
Fonts	Drag and drop files into the list, or click Add Fonts to browse to and add new fonts to your component. InstallShield reads the font title from the font file. If no title is specified within the font file, click on the font title to specify a title in the format FontTitle (FontType)—for example, Marlett (TrueType).
Add Fonts	Click Add Fonts to browse to new font files.
Remove Fonts	Select a font in the list and click Remove Fonts to permanently remove the selected font from the component.

Summary Panel

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *DIM*
- *Merge Module*
- *MSI Database*
- *MSM Database*

If you launched the wizard by right-clicking on a feature, the new component is associated with that feature. Otherwise, you must open the Setup Design view, right-click on one or more features, and select Insert Components to associate your new Font component with a feature.

Note *• Components that are not associated with a feature are displayed with the orphaned component icon.*

When the font's feature is selected for installation, it is automatically registered by Windows Installer.

All Font components are installed by default to the location stored in the FontsFolder property. You can edit the component's Destination Folder or set additional properties by editing the component's property sheet.

Note *• If you would like a font to remain on the system even after the program has been uninstalled, be sure to set the Font component's Permanent property to Yes.*

Unlike the other component types offered in the wizard, there is no corresponding advanced setting for Fonts components. The setting that distinguishes a Fonts component and tells Windows Installer to install it differently is the file's Font Title property.

Component Wizard General Summary Panel

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

This panel appears at the end of the wizard's component creation process. For help with a specific component type, see the following information.

- Setup Best Practices Option

- COM Server Component Type
- Install Services Component Type
- Control Service Component Type
- Fonts Component Type



Note *• Components that are not associated with a feature are displayed with the orphaned component icon.*

Convert Source Paths Wizard

With the Convert Source Paths Wizard, you can convert existing hard-coded paths to path variables, thereby enhancing the portability of your setup project.



Task ***To launch the Convert Source Paths Wizard:***

On the **Project** menu, click **Convert Source Paths**.

Welcome Panel

The Convert Source Paths Wizard allows you to convert your setup's hard-coded paths to path variables. The wizard scans your project for any use of hard-coded paths and then replaces the paths with standard path variables.

If you have files that evaluate to existing path variables, the wizard automatically converts these paths without providing the option of selecting a new variable. For example, if you have a file in the Program Files folder, this path is automatically converted to the predefined path variable <ProgramFilesFolder>.

Search Panel

The Search panel displays the search status. At this time the wizard is scanning the current project for any absolute paths. If your files evaluate to existing path variables, these files are converted automatically.

Search Results and Recommendations Panel

The Search Results and Recommendations panel lists all the path variable options for the files and links in your project that are not located in directories that already have path variables. For example, if your file is `C:\App Files\Executable\App.exe`, you are given the option of creating a path variable called Executable that equates to that path.

Panel Options

Path Variable Recommendation

Indicate the variables that you want to create by selecting the corresponding boxes. If you do not want to create one of the suggested variables, clear the check box next to it.

Rename

Select the variable that you want to rename and click the Rename button. Type the new variable name in the Path Variable Recommendation column.

Deselect All

Click the Deselect All button to clear the check boxes next to all the corresponding suggested path variables.

Select All

Click the Select All button to select all the path variable suggestions. When you click Apply, these variables are created and added to your project.

Updating Your Project Panel

This panel displays the wizard's progress in updating the hard-coded paths in your project to path variables.

Update Completed Panel

This panel signifies the end of the Convert Source Paths Wizard. Click the Finish button to return to the IDE.

Create New QuickPatch Wizard



Project • *This information applies to QuickPatch projects.*

A QuickPatch project is recommended for installation authors who want to ship small, single updates to their users. QuickPatch authoring provides a simple alternative to creating a patch in the Patch Design view, even though it provides less customization. Fundamentally, both patch creation methods produce the same deliverable types: .msp and .exe files.

To create a QuickPatch project for an existing .msi file or an existing QuickPatch, use the Create New QuickPatch Wizard.



Caution • *Creating a QuickPatch project for an earlier QuickPatch without first building the earlier QuickPatch project may cause unexpected behavior.*



Task ***To open the Create New QuickPatch Wizard:***

Create a new project in InstallShield and select QuickPatch Project as the project type.

The QuickPatch you create using this method can be used to patch an existing .msi file or an existing QuickPatch.



Tip • *You can use the Create New QuickPatch Wizard to create a QuickPatch project for an existing QuickPatch. As an alternative, you can open the latest QuickPatch project in InstallShield and specify that you want to create a*

QuickPatch project for the current QuickPatch. For more information, see Creating a QuickPatch Project for an Existing QuickPatch.

Welcome Panel



***Project** • This information applies to QuickPatch projects.*

A QuickPatch project is recommended for installation authors who want to ship small, single updates to their users. QuickPatch authoring provides an alternative to creating a patch in the Patch Design view, even though it provides less customization. Fundamentally, both patch creation methods produce the same deliverable types: .msp and .exe files.

To create a QuickPatch project for an existing .msi file or an existing QuickPatch, use the Create New QuickPatch Wizard.



***Note** • If you click Cancel or Exit in the wizard, QuickPatch project creation will be incomplete, and the QuickPatch project will not open in InstallShield.*

QuickPatch Project Base Panel



***Project** • This information applies to QuickPatch projects.*

The QuickPatch Project Base panel is where you specify whether you would like to base your new QuickPatch project on an existing .msi file or on an existing QuickPatch project.

Panel Options

Table 12-20 • QuickPatch Project Base Panel Options

Option	Description
Based on an MSI package	Select this option to build a QuickPatch project for an existing .msi file. Select this option if this is the first QuickPatch created for the application.
Based an existing QuickPatch project	Select this option to build a QuickPatch project for an existing QuickPatch. This option creates a project that can patch existing QuickPatch projects as well as the original .msi file. ***Note*** • *Attempting to create a QuickPatch of an already existing QuickPatch project without first building the existing QuickPatch causes unexpected behavior.*

If you select the Based on an MSI package option and click Next, the Original Setup Package panel opens. If you select the Based an existing QuickPatch project option and click Next, the Location of Existing QuickPatch panel opens.

Original Setup Package Panel

Project • *This information applies to QuickPatch projects.*

If you are building a QuickPatch project for an existing Windows Installer installation, use the Original Setup Package panel to specify the location of the .msi file. The original installation is the base installation for the new patch. The Create New QuickPatch Wizard creates a patch that can update this installation.

If the original installation is compressed, InstallShield creates an uncompressed image of that installation. The Original Setup Package panel enables you to specify the path for the uncompressed base image.

Note • *Once your project has been created and opened in InstallShield, there is no way to change the path to the base installation. Therefore, if you want to patch a different installation, you must run through the Create New QuickPatch Wizard again to create a new QuickPatch project.*

Panel Options

Table 12-21 • Original Setup Package Panel Options

Option	Description
Original Setup	Specify the location of the original/base installation or click the Browse button. The Create New QuickPatch Wizard creates a project that can validate this installation. If your original installation is uncompressed, click Finish to open your QuickPatch project in InstallShield.
Decompress Path	 ***Note*** • *This option is not available if the original installation is uncompressed.Specify a path that should contain the decompressed image. If you do not specify a path, the following location is used as the default: <ISProjectDataFolder>\BaseImage.*

Click Finish to close the Create New QuickPatch Wizard and open your new QuickPatch project in InstallShield.



Caution • *You cannot decompress installations that span multiple disks from a local or network drive. You will need to find an alternate method of decompressing the installation.*

Location of Existing QuickPatch Panel



Project • *This information applies to QuickPatch projects.*

If you are building a QuickPatch project for an existing QuickPatch, use the Location of Existing QuickPatch panel to specify its location. The QuickPatch that is created can be used to update the original installation, the existing QuickPatch project that you specified, and any selected intermediate QuickPatch projects.



Note • *Once your project has been created and opened in InstallShield, there is no way to change the path to the existing QuickPatch project. Therefore, if you want to patch a different QuickPatch project, you must run through the Create New QuickPatch Wizard again to create a new QuickPatch project.*

Panel Options

Existing QuickPatch Project

Specify the location of the QuickPatch project (.ism file) that you would like to patch or click the Browse button.

Create Table Wizard



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

The Create Table Wizard is launched when you are adding a table through the Direct Editor. This wizard collects information about your custom table, and then adds it to the table list.



Task ***To launch the Create Table Wizard:***

1. In the View List under **Additional Tools**, click **Direct Editor**.
2. Right-click the **Table** explorer and click **Add Table**.

The Create Table Wizard consists of the following panels:

- Welcome
- Column Properties
- Summary

Welcome Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

The Welcome panel provides an introduction to the Create Table Wizard.

Task ***To begin creating a table:***

1. In the table name box, enter a name.

   Windows Logo Guideline • *The Windows logo program requires that you avoid naming a custom table with the* ***MSI*** *(in uppercase, lowercase, or mixed-case letters). The MSI prefix is reserved for future use in new standard properties and tables.*

2. Click the **Add Column** button to proceed to next panel in the wizard.

Column Properties Panel

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

On the Column Properties panel, you configure information for your custom table columns. The following properties are configurable:

Table 12-22 • Column Properties Panel Options

Property	Description
Column Name	Provide a name for the column.
Column Description	Type a description for the column.
Column Type	The available types are Binary, Long, Short, and String.
Nullable Check Box	Select this check box if this column's values are nullable.
Primary Key Check Box	Select this check box to make this column the table's primary key.
Localizable String Check Box	Select this check box if the string is localizable.
String Length	Provide a maximum length for string columns. Enter 0 if the string can be any length.
Min Value	For numeric columns, specify a minimum value.

Table 12-22 • Column Properties Panel Options (cont.)

Property	Description
Max Value	For numeric columns, specify a maximum value.
External Key Table	If this column references an external table, select it from this list
External Key Column	Select the column referenced in an external table.

Click Add Another Column to define additional columns, or click Done with Columns to proceed to the Summary panel.

Summary Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

In the Summary panel, you can review your new table's settings before you add it to your project. If you need to change any of the settings, click Back until you see the appropriate panel, and make the necessary changes.

Click Finish to add the custom table.

Custom Action Wizard



Project • *The Custom Action Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Custom Action Wizard enables you to add custom actions to your installation. These actions can call a .dll file function; launch an executable file (.exe); call a public method in a managed assembly; run VBScript, JScript, or InstallScript code; display an error message; set a directory or a property; and more.

Task ***To launch the Custom Action Wizard:***

1. In the View List under **Behavior and Logic**, click **Custom Actions and Sequences** (in Basic MSI, InstallScript MSI, MSI Database, and Transform projects) or **Custom Actions** (in DIM, Merge Module, and MSM Database projects).
2. Right-click the **Custom Actions** explorer and click **Custom Action Wizard**.

The Custom Action Wizard consists of the following panels:

- Welcome
- Basic Information
- Action Type
- Function Definition
- Action Parameters
- Managed Method Definition
- In-Sequence Scripts
- Additional Options
- Respond Options
- Insert into Sequence
- Summary

Windows Logo *• If you are applying for the Windows logo, any custom actions in your installation must follow best practices guidelines for custom action creation. You can use the InstallShield validation suites and the Full MSI Validation Suite to verify whether your installation package meets most of the custom action–related logo requirements. However, some of the requirements cannot be verified through the validation suite. To learn more, see Guidelines for Creating Custom Actions that Meet Requirements of the Windows Logo Program.*

Welcome Panel

Project *• The Custom Action Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*

- *MSM Database*
- *Transform*

The Welcome panel provides a brief introduction to the Custom Action Wizard. The wizard helps you build a custom action that will perform tasks not inherently supported by Windows Installer.



Windows Logo • *If you are applying for the Windows logo, any custom actions in your installation must follow best practices guidelines for custom action creation. You can use the InstallShield validation suites and the Full MSI Validation Suite to verify whether your installation package meets most of the custom action–related logo requirements. However, some of the requirements cannot be verified through the validation suite. To learn more, see Guidelines for Creating Custom Actions that Meet Requirements of the Windows Logo Program.*

Basic Information Panel



Project • *The Custom Action Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Basic Information panel lets you enter the name of your custom action and a description. This information is used only for your reference and is not displayed to the end user. The following settings are available on this panel.

Table 12-23 • Basic Information Panel Settings

Setting	Description
Name	Enter the name of your custom action. The name can contain only letters, numbers, underscores, or periods, and it must begin with a letter or an underscore.
Comment	Optionally enter comments about your action in this box. This information is for your reference only and is not displayed to the end user.

Action Type Panel



Project • *The Custom Action Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Action Type panel lets you specify the type of custom action that you want to create and where the code for that action is stored.

Panel Options

Type

The Type list provides options for several different ways of executing your custom action. For example, your custom action can call an executable file, run VBScript code, or call a function from a .dll file. The following options are available in this list:

Table 12-24 • Available Types of Custom Actions

Type	Project Type	Description
Call a function in a standard dynamic-link library	Basic MSI, InstallScript MSI	This custom action calls a function in a .dll file. Specify the function name, parameters, and return value in the Function Definition panel. ***Note*** *• If you call a function in a standard .dll file that is stored in the* ***Binary*** *table, you cannot use deferred or rollback execution for the action's In-Script Execution setting.* *If you call a function in a standard .dll file that is installed with the product and the action is scheduled as deferred (for the In-Script Execution setting), the .dll file that you are calling must be the component's key file.*
Call a function in a Windows Installer dynamic-link library	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action calls a function from a .dll file written specifically for Windows Installer. The .dll file's entry point must have a predefined parameter and return value.
Call a public method in a managed assembly	Basic MSI, DIM, InstallScript MSI, Merge Module	This custom action calls a public method in a managed assembly that is written in managed code such as Visual Basic .NET or C#. For more information, see Calling a Public Method in a Managed Assembly.

Table 12-24 • Available Types of Custom Actions (cont.)

Type	Project Type	Description
Display error message and abort	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action displays a specified error message, returns failure, and ends the installation.
Launch an executable	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action launches an executable file.
Launch another .msi package	Basic MSI, InstallScript MSI, MSI Database, Transform	Also known as a nested installation, this type of custom action launches a second .msi package during your installation. ***Important*** • *Nested installations is a deprecated feature of the Windows Installer. Applications installed with nested installations sometimes fail because they are difficult for end users to service correctly. Microsoft Corporation recommends that you avoid using nested installations and nested-installation custom actions to install products that are intended to be released to the public. To learn more, see Concurrent Installations in the Windows Installer Help Library.*
Run 64-bit JScript code	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action runs 64-bit JScript code instead of a compiled language such as C or Visual Basic.
Run PowerShell code	Basic MSI, InstallScript MSI	This custom action runs a PowerShell script (.ps1). For target system requirements, as well as other information about this type of custom action, see Calling a PowerShell Custom Action.

Table 12-24 • Available Types of Custom Actions (cont.)

Type	Project Type	Description
Run 64-bit VBScript code	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action runs 64-bit VBScript code instead of a compiled language such as C or Visual Basic.
Run InstallScript code	Basic MSI, InstallScript MSI, Merge Module	This custom action calls a function from your InstallScript code.
Run JScript code	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action runs JScript code instead of a compiled language such as C or Visual Basic.
Run VBScript code	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action runs VBScript code instead of a compiled language such as C or Visual Basic.
Set a property	Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action sets a property in the **Property** table. This is useful if you need to get information from the end user and store it so that it can be used later in your installation.
Set a directory	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This custom action sets a directory in the **Directory** table at run time. For example, if you want to create a temp directory that is a subdirectory of the installation directory selected by the end user, you can use this option to set that new temp directory for use later in the installation.

Table 12-24 • Available Types of Custom Actions (cont.)

Type	Project Type	Description
Terminate a process by identifier	Basic MSI, InstallScript MSI	This custom action kills a process that has a specific process identifier (PID). ***Important*** • *When you create this type of custom action, you also need to define a property that identifies the PID that you want to stop. To learn more see Calling a Kill-Process Custom Action.*
Terminate a process by name	Basic MSI, InstallScript MSI	This custom action kills a process that has a specific executable file name. ***Important*** • *When you create this type of custom action, you also need to define a property that identifies the name of the process that you want to stop. To learn more see Calling a Kill-Process Custom Action.*

Location

The code that your custom action calls can be stored in one of several places. Select the location where the installer should look for that code. Note that some custom action types do not support some of the locations. In addition, the Location setting is not applicable for the error custom action.

Table 12-25 • Possible Locations for Storing Custom Actions

Location	Custom Action Type	Description
An application that is advertised or already installed	Launch another .msi package	Select this location to uninstall any Windows Installer applications that are currently installed on the target system—for example, applications that you previously installed with a custom action. You should have this custom action launch only on uninstallation of your main installation.
Destination machine search path	Standard DLL	When you call a function in a standard .dll file, the file can be present in the target system's search path.
Included within your main setup	Launch another .msi package	Select this location if the .msi file that you want to launch is stored within your main installation package.

Table 12-25 • Possible Locations for Storing Custom Actions (cont.)

Location	Custom Action Type	Description
Installed with the product	Standard DLL, MSI DLL, Managed code, Executable file, PowerShell code, VBScript, 64-bit VBScript, JScript, 64-bit JScript	Select this location to call code from a file or executable file that is going to be installed on the target system.
Stored directly in the custom action	VBScript, 64-bit VBScript, JScript, 64-bit JScript	Select this location to enter code directly into the wizard, without having to associate a file.
Stored in the Binary table	Standard DLL, MSI DLL, Managed code, Executable file, PowerShell code, VBScript, 64-bit VBScript, JScript, 64-bit JScript	Select this location to have your code base stored in the **Binary** table. This option is useful if you do not want the file to be installed on the target system. ***Note*** • *For the managed-code type of custom action, InstallShield creates a C++ Windows Installer wrapper DLL for your .NET assembly at build time. The wrapper DLL includes your assembly, as well as the information that is required to mediate, load, and run the assembly. InstallShield stores the wrapper DLL in the* ***Binary*** *table.*
Stored in the Directory table	Executable file	Select this location to reference a path—without the file name—that is stored in the **Directory** table.
Stored in the Property table	Managed code, Executable file, VBScript, 64-bit VBScript, JScript, 64-bit JScript	Select this location to reference a full path to the executable that you want to launch. The path is stored in the **Property** table.
Stored on the source media	Launch another .msi package	Select this location if the .msi file that you want to launch is stored on the same media as your main installation package.

Function Definition Panel

The Function Definition panel is where you specify the parameters for the entry-point function in your standard .dll file.

This panel is displayed only if you selected **Call a function in a standard dynamic-link library** in the Action Type panel of the Custom Action Wizard.

Note • *The function must use the __stdcall calling convention. Functions with more than one parameter will not work properly if this convention is not used.*

Panel Options

Name

Specify the name of the function that you want to call. The .dll file can have a single entry point for each custom action.

Arguments

Build the list of arguments—in order—that you want to pass to the function.

First, click the last row of the Arguments list to add a new argument. Next, complete the Type, Source, and Value columns for each argument.

Type

In this list, select the data type of the parameter. Note that there are two special cases for argument types, and these cases are identified below.

Task ***To initialize a pointer to null:***

1. In the **Type** list, select **POINTER**.
2. In the **Source** list, select **Constant**.
3. In the **Value** list, select **NULL**. Do not enter 0 (zero) instead of NULL, because it is interpreted as a pointer pointing to the number 0.

Task ***To set a handle to the .msi database or the Windows Installer window:***

1. In the **Type** list, select **HANDLE**.
2. In the **Source** list, select **Constant**.
3. In the **Value** list, select **MsiHandle** or **MsiWindowHandle**, depending whether you want a handle to the .msi database or the Windows Installer window.

Source

Indicate how you want to pass the data to the function. The following options are available in this list.

Table 12-26 • Options for Passing Data to a Function

Option	Description
Constant	The value that you want to pass is stored as a literal in your installation project. After you select Constant for the argument's source, enter its definition in the Value column. This "constant" does not require a name or identifier. Do not enclose string literals in quotation marks.
in Property	Select in Property for the source to specify the name of a Windows Installer property whose value will be passed to the function. This type of argument is suitable for a ByVal parameter.
inout Property	This source type is similar to a ByRef parameter. Select inout Property for the source to specify the name of a Windows Installer property whose value will be passed to the function and can be modified by the function.
out Property	An out property serves as a placeholder for a value that can be set by the function. No value is passed to the function, but the function requires the name of a property whose value it can modify.

When you select a property for the argument's source, you must specify the name of the property in the Value column.

Value

The value can be one of two types:

- A number or string literal that you enter when the argument's source is a constant
- A Windows Installer property when the argument's source is a property



Note • *When you set the Type list to HANDLE and the Source list to Constant, the Value column contains two options: MsiHandle and MsiWindowHandle. These constants can be used to get a handle to either the .msi database or the Windows Installer window.*

When the source is a property, the Value list provides the name of every property in the Property Manager. You can select an existing property or enter a new name, in which case the new property is added to the Property Manager.

Remember that a property is merely a container for a value. If your parameter stores its value in an in property or inout property, ensure that you specify a value for the property in the Property Manager.

(Context Menu)

The context menu provides you with options for working with your arguments. To access the context menu, right-click an argument in the Arguments grid. The context menu options are described below.

Table 12-27 • Context Menu Options

Option	Description
Add	Click Add to add an argument to the Arguments grid.
Remove	Click Remove to delete the argument.
Move Up	The arguments must be in the precise order in which the function expects to receive them. Click Move Up to place the argument higher in the list.
Move Down	Click Move Down to place the argument lower in the list.

Return Type

Select the data type of the function's return value. If you do not need to check the return value, you can accept void.

Return Property

Select the name of a property whose value will be set to the function's return value. If you are going to check the return value elsewhere in your installation, you may want to initialize the return property's value.



***Note** • The property cannot be set to the function's return value unless the action is configured for immediate execution. Thus, if the action is scheduled for deferred, rollback, or commit execution, the Return Property setting is ignored at run time.*

Silent mode

If you want the installation to suppress a warning message whenever the custom action fails to load the .dll file and call the function, select this check box.

Action Parameters Panel

The Action Parameters panel is where you specify the actual file that you call in your custom action. For example, if you call an executable file, you can browse to that file and add any command-line options.

This panel is not displayed if your custom action is running JScript or VBScript code that is stored directly within the action.

Panel Options

Source

This box behaves differently depending on the options you selected for the Type and Location lists on the Action Type panel. The Source field is disabled if it does not apply to the type of custom action you are adding (such as a New Kill Process custom action). The following table shows the source values that you can use, depending on the type and location of each custom action.

Table 12-28 • Source Values for Custom Actions

Custom Action Type	Instructions for Setting the Source
Call a function in a standard DLL	Set the source for this custom action type according to its location. • **Destination machine search path**—Browse to the location of the .dll file. The wizard uses only the file name and not the path. • **Installed with the product**—Click Browse to open the Browse for File dialog box. In the list of destination folders, select the folder in which the .dll file is located. Then, in the list of files, select the standard .dll file. • **Stored in the Binary table**—Browse to the location of the .dll file.
Call a function in a Windows Installer DLL	Set the source for this custom action type according to its location. • **Installed with the product**—Click Browse to open the Browse for File dialog box. In the list of destination folders, select the folder in which the .dll file is located. Then, in the list of files, select the Windows Installer .dll file. • **Stored in the Binary table**—Browse to the location of the .dll file.
Call a public method in a managed assembly	Set the source for this custom action type according to its location. • **Installed with the product**—Click Browse to open the Browse for File dialog box. In the list of destination folders, select the folder in which the .NET assembly is located. Then, in the list of files, select the assembly. • **Stored in the Binary table**—Browse to the location of the .NET assembly. • **Path referencing a Property**—Select a Windows Installer property in the list, type the name of a new property, or type the name of a directory in the **Directory** table. The property or directory name must be enclosed within square brackets ([]). You can add a string after the property if appropriate. The resulting path should indicate the location of the .NET assembly.
Display error message and abort	The Source setting is disabled for the error custom action, since it is not applicable to this type of custom action.

Table 12-28 • Source Values for Custom Actions (cont.)

Custom Action Type	Instructions for Setting the Source
Launch an executable	Set the source for this custom action type according to its location. • **Installed with the product**—Click Browse to open the Browse for File dialog box. In the list of destination folders, select the folder in which the executable file is located. Then, in the list of files, select the executable file. • **Stored in the Binary table**—Browse to the location of the .exe file. • **Stored in the Property table**—Select a Windows Installer property in the list. If you specify the name of a new property, you must later open the Property Manager, enter the property, and specify a value that points to the .exe file. • **Stored in the Directory table**—Select one of the standard folders in the list, or enter the name of a new entry into the **Directory** table—which you can later edit in the Direct Editor. The directory that you specify is the working directory for the target that you specify in the Target box.
Launch another .msi package	Set the source for this custom action type according to its location. (For more information about custom actions that launch another .msi package, see Nested Installations.) • **Included within your main setup**—Browse to the location of the "child" .msi file. The Custom Action Wizard automatically creates the appropriate entries in the Binary table. • **Stored on the source media**—Browse to the location of the "child" .msi file. The Custom Action Wizard automatically adds this file to the Disk1 folder in the Support Files view. • When you build a release, InstallShield copies the .msi file (but not its uncompressed application and support files) to the release location. • **An application that is advertised or already installed**—Browse to the location of the "child" .msi file. The Custom Action Wizard extracts the product code from the .msi package that you select.
Run InstallScript code	You cannot change the location of the InstallScript file. For an InstallScript custom action, you must select the name of the entry-point function in the list.

Table 12-28 • Source Values for Custom Actions (cont.)

Custom Action Type	Instructions for Setting the Source
Run JScript or 64-bit JScript code	Set the source for this custom action type according to its location. • **Installed with the product**—Click Browse to open the Browse for File dialog box. In the list of destination folders, select the folder in which the .js file is located. Then, in the list of files, select the .js file. • **Stored in the Binary table**—Browse to the location of the JScript source file. • **Stored in the Property table**—Select one of the properties in the list. If you specify a new property, you must later open the Property Manager, enter the property name, and add the JScript code as the value of the property. • **Stored directly in the custom action**—You can use the In-Sequence Scripts panel, which provides a text editor for your JScript code.  ***Note*** • *The In-Sequence Scripts panel is provided for backward compatibility only. A superior script editor is available in the Script tab of the Custom Actions and Sequences view (and the Custom Actions view).*
Run PowerShell code	Set the source for this custom action type according to its location. • **Installed with the product**—Click Browse to open the Browse for File dialog box. In the list of destination folders, select the folder in which the .ps1 file is located. Then, in the list of files, select the .ps1 file. • **Stored in the Binary table**—Browse to the location of the PowerShell file (.ps1).

Table 12-28 • Source Values for Custom Actions (cont.)

Custom Action Type	Instructions for Setting the Source
Run VBScript or 64-bit VBScript code	Set the source for this custom action type according to its location. • **Installed with the product**—Click Browse to open the Browse for File dialog box. In the list of destination folders, select the folder in which the .vbs file is located. Then, in the list of files, select the .vbs file. • **Stored in the Binary table**—Browse to the location of the VBScript source file. • **Stored in the Property table**—Select one of the properties in the list. If you specify a new property, you must later open the Property Manager, enter the property name, and add the VBScript code as the value of the property. • **Stored directly in the custom action**—You can use the In-Sequence Scripts panel, which provides a text editor for your VBScript code.  ***Note*** • *The In-Sequence Scripts panel is provided for backward compatibility only. A superior script editor is available in the Script tab of the Custom Actions and Sequences view (and the Custom Actions view).*
Set a directory	You cannot set the location when setting a Directory. You must select the name of a folder in the list, or enter a new name and specify it in the **Directory** table (available in the Direct Editor).
Set a property	You cannot set the location when setting a property. You must select the name of the property in the list, or enter a new name and specify it in the Property Manager.
Set a property	You cannot set the location when setting a property. You must select the name of the property in the list, or enter a new name and specify it in the Property Manager.
Set a property	You cannot set the location when setting a property. You must select the name of the property in the list, or enter a new name and specify it in the Property Manager.

Target

The value for the target depends on the custom action type.

Table 12-29 • Target Options

Custom Action Type	Instructions for Setting the Target
Call a function in a standard DLL	For the standard DLL type of custom action, the Target box is read-only. It displays the function definition that is specified on the Function Definition panel.

Table 12-29 • Target Options (cont.)

Custom Action Type	Instructions for Setting the Target
Call a function in a Windows Installer DLL	Use this box to enter the name of the function that you want to call. You do not need brackets or any other special formatting, but note that the function name is case-sensitive. A Windows Installer .dll function has a predefined parameter and return type. For more information, see Passing Parameters to a DLL File Function in a Custom Action.
Call a public method in a managed assembly	For a managed assembly type of custom action, the Target box is not applicable. The Managed Method Definition panel, which is the next panel in the wizard for this type of custom action, is where you identify the method that you want to be invoked.
Display error message and abort	Type the error message text that should be displayed when the custom action's conditions are met on the target system. As an alternative, you can type a property name whose value contains the error text. The property name must be enclosed within square brackets ([]).
Launch an executable	If you are launching an executable file, you can enter command-line arguments in this box. For example, if you want to launch a readme file that is installed with you installation, you should enter Notepad.exe [INSTALLDIR]Readme.txt.
Launch another .msi package	Specify any public properties that you would like to pass to the .msi package.
Run InstallScript code	The target value is not applicable for InstallScript custom actions.
Run PowerShell code	The target value is not applicable for PowerShell custom actions.
Run JScript/64-bit JScript or VBScript/64-bit VBScript code	Enter the function name that you want to call.
Set a property or directory	Enter a formatted text string that evaluates to the new value of the property that you selected or created. See the Windows Installer Library for more information on formatted text strings.
Terminate a process by identifier	Enter the PIDs of the processes that you wish to terminate for the Kill Process custom action. To specify more than one process, use a semicolon (;) to separate each PID.
Terminate a process by name	Enter the executable file names of the processes that you wish to terminate that you wish to terminate for the Kill Process custom action. To specify more than one process, use a semicolon (;) to separate each file name.

Managed Method Definition Panel

The Managed Method Definition panel is where you specify the public class method in the managed assembly. The Custom Action Wizard displays this panel only if you selected **Call a public method in a managed assembly** in the Action Type panel.

The Managed Method Definition panel has the following settings.

Table 12-30 • Settings on the Managed Method Definition Panel

Setting	Description
Class	Enter the public class with which the method is associated. As an alternative, click the Browse button to display the Method Browser dialog box. This dialog box lets you select the public method from the list of public classes that are available in the managed assembly that you selected on the Action Parameters panel. ***Note*** • *The browse functionality is available only if the .NET Framework is installed. In addition, it is available only if the location of the managed assembly is set to the* ***Binary*** *table or installed with the product—not if the assembly's path references a property or a directory in the* ***Directory*** *table.* *Also note that the browse functionality is available for only 32-bit assemblies and for Microsoft intermediate language (MSIL) files. It is not available for 64-bit assemblies.*
Method	Enter the public method that you want your installation to call. As an alternative, click the Browse button to display the Method Browser dialog box. This dialog box lets you select the public method from the list of public classes that are available in the managed assembly that you selected on the Action Parameters panel. ***Note*** • *The browse functionality is available only if the .NET Framework is installed. In addition, it is available only if the location of the managed assembly is set to the* ***Binary*** *table or installed with the product—not if the assembly's path references a property or a directory in the* ***Directory*** *table.* *Also note that the browse functionality is available for only 32-bit assemblies and for Microsoft intermediate language (MSIL) files. It is not available for 64-bit assemblies.*

Table 12-30 • Settings on the Managed Method Definition Panel (cont.)

Setting	Description
Use custom method signature	To use the default method signature, clear this check box. For the default method signature, the installation calls the method with an MsiHandle parameter if the signature has one or more void parameters, or if it has a single MsiHandle parameter. To use a custom method signature, select this check box. Then specify the arguments and return property as needed. If you specified a class and a method by clicking the Browse button, InstallShield automatically adds any associated parameters. Specify the value for each parameter that you want to pass to the method. For detailed information about specifying the method's signature, see Specifying the Signature for a Managed Method in an Assembly Custom Action.
Arguments	If you are using a custom method signature, specify the list of arguments that you want to pass to the selected method. If you specified a class and a method by clicking the Browse button, InstallShield automatically adds any associated parameters to the grid in this area. To use a property as an argument, click the value field for the parameter, and then select the property from the list. The context menu provides you with options for working with arguments. To access the context menu, right-click an argument in the Arguments grid. The available context menu commands are: • **Add**—Add an argument to the grid. • **Remove**—Delete an argument from the grid. This grid is available only if the **Use custom method signature check box** is selected. For more information about specifying the arguments, see Specifying the Signature for a Managed Method in an Assembly Custom Action.
Return Property	Select the name of the property whose value will be set to the method's return value. This list is available only if the **Use custom method signature check box** is selected. For more information about specifying the return property, see Specifying the Signature for a Managed Method in an Assembly Custom Action.

In-Sequence Scripts Panel



Note *• The In-Sequence Scripts panel is provided for backward compatibility only. A superior script editor is available in the Script tab of the Custom Action and Sequences view (and the Custom Actions view).*

If your custom action calls JScript or VBScript code that is stored directly in the custom action, you can enter the script in the box that is displayed. Type your code into the text box and click Next to continue. No validation is performed on your code to check syntax.

Additional Options Panel

The Additional Options panel lets you specify how the custom action thread should be processed and whether the custom action should be run only during patch uninstallation.

The Additional Options panel has the following settings.

Table 12-31 • Settings on the Additional Options Panel

Setting	Description
Return Processing	Specify how the custom action thread should be processed. Valid options for the Return Processing list are: • Asynchronous (No wait for completion) • Asynchronous (Wait for exit code) • Synchronous (Check exit code) • Synchronous (Ignores exit code) These flags are used to specify that the main and custom action threads run synchronously (the installer waits for the custom action thread to complete before resuming the main installation thread) or asynchronously (the installer runs the custom action simultaneously as the main installation continues). Note that some options are not applicable to some types of custom actions. Only options that pertain to the selected type of custom action are available in this list.

Table 12-31 • Settings on the Additional Options Panel (cont.)

Setting	Description
Run During Patch Uninstall	 ***Project*** • *If you are working on a project in Direct Edit mode, this setting is not applicable unless the database schema is a minimum of 405 (for Windows Installer 4.5 or later).* This check box lets you specify whether Windows Installer should run the custom action only when a patch is being uninstalled. You can select this check box for a custom action in an .msi package. You can also select this check box for a new custom action that is added by a patch. However, this check box should not be selected for a custom action that is being added or removed by a patch to an existing custom action. This check box is cleared by default. When Windows Installer 4.5 runs the patch uninstall custom action, it uses the custom action in the patch that is being uninstalled.  ***Note*** • *Windows Installer 4.5 includes support for this setting, but Windows Installer 4.0 and earlier do not. Therefore, if you select Yes for this setting and some target systems may have Windows Installer 4.0 or earlier, add a condition to this custom action to prevent Windows Installer 4.0 and earlier from running it. Otherwise, Windows Installer 4.0 and earlier would call the custom action during the installation, repair, or update of the package.* For the condition, use the **MSIPATCHREMOVE** property. For more information, see MSIPATCHREMOVE in the Windows Installer Help Library.

Respond Options Panel

The Respond Options panel lets you specify how your custom action will respond to the rest of the installation.

Table 12-32 • Respond Options Panel Options

Option	Description
In-Script Execution	Specify when you would like your action to execute. • **Immediate execution**—Select this option to have the action execute when it is encountered in the sequence. • **Immediate execution (Terminal Server Aware)**—Select this option to have the action execute when it is encountered in the sequence. The custom action impersonates the user during per-machine installations on terminal server machines. • **Deferred execution**—Select this option to have your action wait until the script begins to run. • **Rollback execution**—Select this option to set your action to execute only during rollback. If you changed something to the system during the execute sequence using a custom action, you need to undo that change with a rollback custom action. • **Rollback execution (Terminal Server Aware)**—Select this option to set your action to execute only during rollback. If you changed something to the system during the execute sequence using a custom action, you need to undo that change with a rollback custom action. The custom action impersonates the user during per-machine installations on terminal server machines. • **Commit execution**—Select this option to have your action delayed until the installation has successfully completed the installation script. • **Commit execution (Terminal Server Aware)**—Select this option to have your action delayed until the installation has successfully completed the installation script. The custom action impersonates the user during per-machine installations on terminal server machines. • **Deferred execution in system context**—Select this option if your action needs elevated system privileges for performing tasks such as editing the registry. The action is deferred until the script begins to run. When the action does run, it runs with elevated system privileges. For more information on this property, see In-Script Execution.  ***Note*** • *If you are calling a function in a standard dynamic-link library that is stored in the* ***Binary*** *table, you cannot use deferred or rollback execution for the action's In-Script Execution property.*

Table 12-32 • Respond Options Panel Options (cont.)

Option	Description
Execution Scheduling	Specify how many times you want the action to run. For example, if an action is listed in both the user interface and execute sequences, it can be set to run both times, or it can run only once. Available options are: • **Always execute**—The action runs every time it is encountered. Therefore, it could run in the user interface sequence and again in the execute sequence. • **Execute only once**—The action runs only once—even if it exists in both the user interface and Execute sequences. The action is skipped in the execute sequence if the user interface sequence has run.  ***Note*** • *The **Execute only once** option does not work for InstallScript MSI projects. If your custom action exists in both the user interface and execute sequences, it is executed twice. This is because—in an InstallScript MSI project—the InstallScript engine executes the user interface sequence and Windows Installer executes the execute sequence.* • **Execute only once per process**—The user interface sequence and the execute sequence run in separate processes. Select this option to force your custom action to launch at least once in the user interface sequence and once in the execute sequence. • **Always execute at least once on the client**—This action executes at least one time on the client. If you selected anything other than Immediate execution in the In-Script Execution setting, the Execution Scheduling setting is not available and is set to Always execute.

Insert into Sequence Panel

In the Insert into Sequence panel, you specify where in the installation sequence you want to insert the custom action that the wizard is creating. You can insert your custom action into one sequence or into multiple sequences.

Panel Options

Sequences

Select where in the Installation User Interface or Installation Execute sequence you want to insert your custom action.

Conditions

Click the Browse button to launch the Condition Builder dialog box, where you can create a condition that will be used to determine whether your custom action is launched.

Summary Panel

In the Summary panel, you can review your custom action's settings before you add it to your installation. If you need to change any of the settings, click Back until you see the appropriate panel, and make the necessary changes.

When your action is complete, click Finish.

You can now insert this custom action into a sequence or specify it as the argument for a dialog's control event.

Database Import Wizard



***Note** • The import database functionality applies to the Microsoft SQL Server Database. Oracle users should refer to the Oracle Web page on Oracle Database Utilities for information on utilities that may work in conjunction with InstallShield.*

The Database Import Wizard helps you import your Microsoft SQL Server Database and generate a SQL script file from the database based on options you, the installation author, determine. The wizard can be launched from the following nodes of the SQL Scripts explorer when you right-click in the SQL Scripts view:

- SQL Servers
- New Connection
- New Script



***Note** • InstallShield supports SQL Server 7.0 and later.*

The following wizard panels are associated with the Database Import Wizard:

- Welcome
- SQL Server
- SQL Database
- Database Tables
- Objects to Script
- Scripting Options
- Advanced Scripting Options
- Summary

Welcome Panel



***Note** • The import database functionality applies to the Microsoft SQL Server Database.*

The Database Import Wizard helps you import your Microsoft SQL Server Database and generate a SQL script file from the database based on options you, the installation author, determine.

The second paragraph of this wizard panel indicates what the wizard will create.

SQL Server Panel



***Note** • The import database functionality applies to the Microsoft SQL Server Database.*

Server Name

Specify a SQL Server from the list of servers below or click Browse to select the location of a server on your network.



***Note** • InstallShield supports SQL Server 7.0 and later.*

Importing a database and browsing for a SQL Server requires ODBC driver version 3.80 or later.

With Windows NT authentication using the network login ID

This is the recommended authentication method. SQL Servers must be set up to work with this method. The currently logged in user and the user's credentials will be used.

With Server authentication using the login ID and password below

If you select this option, you will have to enter the login ID and set the password for a particular SQL Server.

SQL Database Panel



***Note** • The import database functionality applies to the Microsoft SQL Server Database.*

Database Name

Choose a database found on the SQL Server you selected in the SQL Server panel. A script file will be generated for the database you select in the Database Name drop down list.

Script Display Name

Type a valid identifier for the script file which will be generated from the SQL database you selected in the Database Name drop down list. The script display name is what is displayed in the explorer in the SQL Scripts view.



***Note** • When the generated script is executed on a target machine, this database will be automatically created if not found on the target server.*

Database Tables Panel



Note *• The import database functionality applies to the Microsoft SQL Server Database.*

In this panel, determine which tables in the database you want to include or exclude in the generated script file. If you choose to include or exclude certain tables, then from the Available Tables list, click the table(s) you want, and then click the ">" button to mark the tables.



Note *• Selecting the Include All Tables option guarantees that new tables added to the database are imported every time you reimport the database.*

Objects to Script Panel



Note *• The import database functionality applies to the Microsoft SQL Server Database.*

The Objects to Script panel is where you select the objects that you want to import from the database.

Objects

Determine which objects to import from your SQL Database. Many of these objects represent and expose attributes of a single Microsoft SQL Server database object.



Note *• The Records option is unavailable if no tables are selected to be imported in the Database Tables wizard panel.*

Scripting Options Panel



Note *• The import database functionality applies to the Microsoft SQL Server Database.*

The Scripting Options panel in the Database Import Wizard is where you specify whether your script should be compatible with Microsoft SQL Server version 7.0. Other options on this panel let you specify whether descriptive headers, extended properties, and other information should be included in your script. Transact-SQL statements are generated as needed.

General Options

Table 12-33 • General Options

Option	Description
Create Only If Missing	To generate a Transact-SQL statement that creates a component prefixed by a check for existence, select this check box. When the script is executed, a component is created only if a copy of the named component does not exist. ***Note*** • *Currently, this option applies only to tables.*
Drop First If Exists	To generate a Transact-SQL statement that removes a referenced component, select this check box. The script tests for the existence of the component prior to an attempt to remove the component.
Include Descriptive Headers	To include explanatory header text before each Transact-SQL statement in the script, select this check box.
Include Extended Properties	To include extended stored procedures in the SQL scripts that are created, select this check box.
Only Script 7.0 Compatible	To generate a script that is compatible with Microsoft SQL Server version 7.0, select this check box. If you select this check box, the following SQL Server 2000 options are ignored: • Column level collation • User-defined functions • Extended property • INSTEAD OF trigger on tables and views • Indexes on views (indexed views) • Indexes on computed columns • Reference permissions on views • Descending indexes

Table Scripting Options

Table 12-34 • Table Scripting Options

Setting	Description
Script Indexes	To generate a Transact-SQL statement that creates indexes that currently exist for any selected tables, select this check box. ***Note*** • *Selecting this check box is useful only if one or more tables are selected on the Database Tables panel.*
Script Full-Text Indexes	To generate a Transact-SQL statement that creates full-text indexes, select this check box. Selecting this check box is useful only if one or more tables are selected on the Database Tables panel.
Script Triggers	To generate a Transact-SQL statement that creates triggers that exist for any selected tables, select this check box. ***Note*** • *Selecting this check box is useful only if one or more tables are selected on the Database Tables panel.*
Script Check Constraints	To generate a Transact-SQL statement that creates check constraints that exist for any selected tables, select this check box. ***Note*** • *Selecting this check box is useful only if one or more tables are selected on the Database Tables panel.*
Script Foreign Keys	To generate a Transact-SQL statement that creates FOREIGN keys that exist for any selected tables, select this check box. ***Note*** • *Selecting this check box is useful only if one or more tables are selected on the Database Tables panel.*
Script Primary Keys	To generate a Transact-SQL statement that creates PRIMARY keys that exist for any selected tables, select this check box. ***Note*** • *Selecting this check box is useful only if one or more tables are selected on the Database Tables panel.*

Table 12-34 • Table Scripting Options (cont.)

Setting	Description
Script Defaults	To generate a Transact-SQL statement that creates defaults that exist for any selected tables, select this check box. ***Note*** • *Selecting this check box is useful only if one or more tables are selected on the Database Tables panel.*

Advanced Scripting Options Panel

Note • *The import database functionality applies to the Microsoft SQL Server Database.*

The Advanced Scripting Options panel lets you specify security scripting options, as well as the file format.

Security Scripting Options

Table 12-35 • Security Scripting Options

Setting	Description
Script database	Generate a Transact-SQL statement to create a script of the existing database schema.
Script database users and database roles	Generate a Transact-SQL statement to create all users and roles that have access to the database.
Script SQL Server logins (Windows NT and SQL Server logins)	Generate a Transact-SQL statement to create all logins that currently have access to the server.
Script object-level permissions	Generate a Transact-SQL statement to create all grant, revoke, and deny permissions that currently exist for each object selected on the Objects to Script panel.

File Format

Table 12-36 • File Format Options

Option	Description
Windows Text (ANSI)	This is the default option. If you want the script to be in ANSI format, select this option.
International Text (Unicode)	International databases can use the Unicode file format for the script, if necessary. If you select this option, Unicode BOM encoding for your SQL script.



***Note** • Consult your SQL-DMO help reference for more detailed information about script formatting and file formats.*

Summary Panel



***Note** • The import database functionality applies to the Microsoft SQL Server Database.*

The Summary panel provides a summary of all the wizard settings.

Check the Regenerate Script at Build option if you want to regenerate the script file each time you build your installation project.



***Note** • The Regenerate Script at Build option will slow down your build since a connection to the server needs to be established, and the database needs to be reimported every time you rebuild your project.*

Click Finish to generate the script file (.sql). Once the script is generated, you can edit it from the script's Script tab in the SQL Scripts view.

Device Driver Wizard



Project *• The Device Driver Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

The Device Driver Wizard simplifies the process of installing device drivers from a Windows Installer–based installation using the Driver Installation Frameworks for Applications (DIFxApp) from Microsoft. The wizard creates the necessary table and entries, custom actions, feature, and components.

For the latest available information on the Driver Installation Frameworks, visit Windows Hardware and Driver Central at http://www.microsoft.com/whdc/.



Task ***To open the Device Driver Wizard, do one of the following:***

- In the **Setup Design** view, right-click a feature and then click **Device Driver Wizard**. The feature that the wizard creates will be a subfeature of the selected feature.
- On the **Project** menu, click **Device Driver Wizard**. The feature that the wizard creates will be a root-level feature.

The following panels are associated with the Device Driver Wizard:

- Welcome
- Device Driver Package
- Device Driver Files
- Device Driver Information
- Project-Wide Device Driver Information
- Summary

Welcome Panel

The Device Driver Wizard simplifies the process of installing device drivers from a Windows Installer–based installation using Microsoft's Driver Install Frameworks for Applications (DIFxApp). The wizard creates the necessary table and entries, custom actions, feature, and components.



Tip *• You can disable the Welcome panel by choosing the Tools menu's Options command and clearing the Options dialog box's* ***User Interface tab's Show Welcome panel in IDE wizards*** *option.*

Device Driver Package Panel

In the Device Driver Package panel, specify the package file (.inf file) for the device driver that you want to add. Type the fully qualified file name of the package, or click the Browse button to navigate to the file.



***Note** • Clicking Next displays a warning message if the specified package file does not specify a security catalog, if the specified security catalog is missing, or if the package file has been modified or corrupted since it was signed. For more information, visit http://www.microsoft.com/whdc/winlogo/drvsign/drvsign.mspx.*

Device Driver Files Panel

The Device Driver Files panel displays the files that the wizard has detected as belonging to the device driver.

Device Driver Information Panel

In the Device Driver Information panel, you can view the device driver type that is specified in your .inf file, and optionally select any of the following runtime options:

Always overwrite any existing device driver

Replace any existing driver for the device with this driver. If this check box is cleared and a driver already exists for the device, your driver is not installed.

Do not show Connect Device to Computer dialog

To prevent the Connect Device to Computer dialog from being displayed during the installation of a device driver if a device matching the driver is not connected to a computer, select this option.

Do not create Add or Remove Programs entry for device driver

If you select this check box, the installation does not create an Add or Remove Programs entry for the device driver. If you clear this check box, the installation does create an Add or Remove Programs entry.

DIFxApp does not support this feature on Windows Vista and later.

Install unsigned driver files and drivers with missing files

By default, DIFxApp does not install unsigned driver packages or driver packages that have missing files. To override this default behavior, select this check box. Selecting this check box enables you to fully test drivers more easily before shipping final signed versions.

Remove binary files associated with driver during uninstall

By default, when DIFxApp uninstalls a driver package, DIFxApp does not remove the binary files that were copied to the system when it installed the driver. To override this default behavior, select this check box.

If you select this check box, DIFxApp removes a binary file from the system only if the binary file is identical to the corresponding binary file in the driver store.



Caution • *Select the* ***Remove binary files associated with driver during uninstall*** *check box only if you can verify that the driver's binary files are not required by any other driver package or application.*

Device Driver Type

The Device Driver Information panel displays a device driver type, which is determined in the following way:

- If the [Version] section of the package file (.inf file) has a DriverPackageType entry whose value is one of those in the following table, the corresponding driver type is displayed.
- Otherwise, the displayed device driver type is Plug and Play driver.

Table 12-37 • DriverPackageType Values and Corresponding Driver Types

DriverPackageType Value	Corresponding Driver Type
ClassFilter	Class filter driver
FileSystem	File system driver
FileSystemFilter	File-system filter driver
KernelModule	Export driver
KernelService	Kernel service driver
Network	Network driver
PlugAndPlay	Plug and Play driver

InstallShield uses Microsoft Windows Driver Install Framework (DIFx) to install drivers. When your device driver is installed, DIFx treats it as being a device driver of the type that is displayed in the Device Driver Information panel.

To change the device driver type, change the value of the DriverPackageType entry in the package file's [Version] section to the desired value from the preceding table, and then re-sign the package file. (For more information on signing package files, visit http://www.microsoft.com/whdc/winlogo/drvsign/drvsign.mspx.

Project-Wide Device Driver Information Panel

The settings that you configure on the Device Driver Information panel are applied to all device drivers in this project. The settings are:

Include all localized installation runtime dialogs

When this check box is selected, the installation uses a custom action runtime .dll that includes dialogs and text for the following languages:

- Arabic (Saudi Arabia)
- Chinese (People's ROC)

- Chinese (Taiwan)
- Czech (Czech Republic)
- Danish (Denmark)
- Dutch (Netherlands)
- English (United States)
- Finnish (Finland)
- French (France)
- German (Germany)
- Greek (Greece)
- Hebrew (Israel)
- Hungarian (Hungary)
- Italian (Italy)
- Japanese (Japan)
- Korean (Korea)
- Norwegian (Bokmål) (Norway)
- Polish (Poland)
- Portuguese (Brazil)
- Portuguese (Portugal)
- Russian (Russia)
- Spanish - Modern Sort (Spain)
- Swedish (Sweden)
- Turkish (Turkey)

If you clear this check box, the installation uses runtime dialogs for English only. The English runtime .dll file is smaller than the localized .dll file, so if space is an issue and you do not need the extra language runtime dialogs, clear this check box.

Device driver machine architecture

This area has several options. Select the appropriate option.

- Device driver targets 32 bit machines
- Device driver targets Itanium 64 bit machines
- Device driver targets AMD 64 bit machines

Summary Panel

The Summary panel displays the options that you have specified in the wizard. Click the Finish button to complete the wizard.

Dialog Wizard

InstallShield provides a Dialog Wizard that enables you to add a new dialog to your installation and configure it by stepping through the wizard panels.



Task ***To launch the Dialog Wizard:***

1. Open the **Dialogs** view.
2. In the **Dialogs** explorer, right-click **All Dialogs** and then click **New Dialog**. The Dialog Wizard launches.
3. Follow the wizard panels.

The following panels are associated with the Dialog Wizard:

- Welcome
- Dialog Template
- User Interface Sequence
- Dialog Position and Condition

Welcome Panel

The Welcome panel welcomes you to the wizard and briefly describes the Dialog Wizard's function. Click Next to begin using the wizard.

Dialog Template Panel

Use the Dialog Template panel to select a default template or an existing dialog on which to base your new dialog.

Panel Options

List of Dialogs

The box in the Dialog Template panel lists dialogs that are in the directory specified in the Dialog Location box on the File Locations tab of the Options dialog box. It may also include dialogs that are stored in a repository. In addition, several default dialog templates are listed:

Table 12-38 • Dialog Template Options

Dialog Template	Description
Blank Dialog	Creates a blank dialog with no dialog controls.  ***Project*** *• In InstallScript and InstallScript MSI projects, this dialog template does not have a hidden control that has a Control Identifier value of 2; the NewScriptBasedDialog and NewSkinnableDialog templates do include this control. To enable end users to cancel the installation by clicking the close button in the upper-right corner of the InstallScript dialog, a custom dialog must have a button control whose Control Identifier property is set to this value. For more information, see Using InstallScript to Implement Custom Dialogs.*
Exterior Wizard Panel	 ***Project*** *• This dialog template is available for Basic MSI projects.* Creates a dialog with a large bitmap along the left side of the dialog—similar to the formatting of the Welcome dialog.
Interior Wizard Panel	 ***Project*** *• This dialog template is available for Basic MSI projects.* Creates a dialog with a small bitmap at the top of the dialog.
Logon Information Panel and Associated Child Dialogs	 ***Project*** *• This dialog template is available for Basic MSI projects.* *For information on adding the Login Information dialogs to Basic MSI, InstallScript, or InstallScript MSI projects, see Adding the Ability to Create or Set an Existing User Account.* Creates a set of dialogs that enable the end user to create or browse for an existing user account in order to access resources that are restricted to others.

Table 12-38 • Dialog Template Options (cont.)

Dialog Template	Description
NewScriptBasedDialog	 ***Project*** • *This dialog template is available for the following project types:* • *InstallScript* • *InstallScript MSI* Creates a new script-based dialog that contains the same controls as those contained in an internal dialog. This dialog contains basic controls, including Back, Next, Cancel, Title, and Subtitle.
NewSkinnableDialog	 ***Project*** • *This dialog template is available for the following project types:* • *InstallScript* • *InstallScript MSI* Creates a new script-based, *skinnable* dialog that contains some of the same controls as those contained in an internal dialog. This dialog contains basic controls—including Back, Next, and Cancel. You should select this option if you plan to use dialog skins.  ***Caution*** • *If you add any controls to this dialog, do not set the control ID to 52. If you do, the dialog skins will not work.*

Show Dialogs in Repository

To display dialogs that are stored in a repository, select the Show Dialogs in Repository check box.

Browse

Click the Browse button if you want to select to a dialog (.isd) file that is not listed.

Let me Insert this dialog into a sequence

Select this check box to automatically insert the new dialog in a user interface sequence. This check box is available only for Basic MSI projects and only for Interior and Exterior dialogs. In InstallScript MSI projects, you need to use InstallScript to schedule the dialog's appearance in the user interface and to process the dialog's controls.



Note • *If this option is not selected in a Basic MSI project, the wizard process ends and the new dialog is added to your project. You need to manually add the dialog to a sequence to display it during installation.*

Basic MSI Projects

If you selected an exterior, interior, or login dialog, click Next to move to the next panel. If you selected a blank dialog, a dialog in your repository, or a dialog in a different location, click Finish to add the dialog to your project.

***Note** • You need to manually insert a blank dialog into a sequence in order to display it during installation.*

InstallScript and InstallScript MSI Projects

Click Finish to add the new dialog. In order for the new dialog to appear in the user interface, you need to add it to your script and use InstallScript to process the dialog controls. You can edit the dialog controls if necessary.

User Interface Sequence Panel

The User Interface panel lets you select the user interface sequence into which you want to insert the new dialog. You can also see which dialogs are currently in the selected sequence and insert the new dialog after a particular existing dialog.

Panel Options

Table 12-39 • User Interface Sequence Panel Options

Option	Description
User Interface Sequence	From the menu, select the sequence into which you want to insert the new dialog.
Dialogs currently in this sequence	Displays the dialogs that occur in the selected sequence. Select the dialog that you want to appear immediately before your new dialog in the sequence.

Dialog Position and Condition Panel

Use the Dialog Position and Condition panel to indicate where you want the dialog to appear in the sequence. This panel also lets you specify any conditions that you want placed on the dialog.

Panel Options

Table 12-40 • Dialog Position and Condition Panel Options

Option	Description
Dialog Position	This pane displays the new dialog's place in the sequence selected in the User Interface Sequence panel. To move the new dialog within the sequence, select it and click Move Up or Move Down.
Condition	Set a condition for the new dialog in this field. Click Build to launch the Condition Builder dialog box.

Click Finish to exit the wizard. InstallShield adds the new dialog to your project and inserts it in the specified sequence.

DirectX Object Wizard

Project *• The DirectX Object Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The DirectX Object Wizard enables you to include the Microsoft DirectX 9c redistributable files in your installation project and to set several options.

Note *• For information about which files are included with the DirectX object, see Including the DirectX 9.0 Object.*

The optional ManagedDX component of DirectX 9c requires that the .NET Framework version 1.1 or later be installed on the system.

The following wizard panels are associated with the DirectX Object Wizard:

- Welcome
- Object Settings
- Summary

For more information about DirectX, visit http://msdn.microsoft.com/directx.

Welcome Panel

Project *• The DirectX Object Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The DirectX Object Wizard enables you to include the Microsoft DirectX 9c redistributable files in your installation project and to set several options.

Note *• For information about which files are included with the DirectX object, see Including the DirectX 9.0 Object.*

The optional ManagedDX component of DirectX 9c requires that the .NET Framework version 1.1 or later be installed on the system.

For more information about DirectX, visit http://msdn.microsoft.com/directx.

Object Settings Panel

Project *• The DirectX Object Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The Object Settings panel enables you to set the following options:

Table 12-41 • Object Settings Panel Options

Option	Description
Place DirectX files in a folder in Disk1	If you select this check box, InstallShield creates a DirectX folder for your installation at build time and places it in the Disk1 folder of your release. If you clear this check box, InstallShield streams the DirectX redistributable files into the .msi file. If you are creating a single-file executable installation, clear this check box so that the files are included in the .msi file. For information on the DirectX files that are included in the object, see Including the DirectX 9.0 Object.
Show the Microsoft DirectX EULA before starting the DirectX 9 installation	By default, the Microsoft DirectX EULA is displayed to the end user before the DirectX 9 installation begins. To prevent the EULA from being displayed, clear this check box.

For more information about DirectX, visit http://msdn.microsoft.com/directx.

Summary Panel

Project • *The DirectX Object Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The Summary panel enables you to review the options that you configured on the Object Settings panel. Click Finish to accept the settings and add the DirectX 9 redistributable files to your project.

Dynamic Scanning Wizard

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Dynamic Scanning Wizard is an easy-to-use tool that monitors your system while an executable file runs. The wizard displays a list of .dll and .ocx files that may be required by the executable file, and it lets you specify whether you want to include each one in your project. The wizard can scan for an executable file that is already included in your project, or you can use the Specify the Executable panel in the wizard to select a new executable file that you want to scan and add to your project prior to the scanning process.

Task ***To launch the Dynamic Scanning Wizard:***

1. In the View List under **Additional Tools**, click **Dependency Scanners**.
2. Click the **Perform Dynamic Scanning** button.

The following panels are associated with the Dynamic Scanning Wizard:

- Welcome
- Filter Files
- Specify the Executable
- Specify Application File
- Launch the Application
- Your Application is Running
- File Selection
- Scan Results
- Completing the Dynamic Scanning Wizard

Welcome Panel

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Dynamic Scanning Wizard provides you with an easy path to add your application's dependency files to your project. Before you begin using the scanner, it is recommended that you add the target executable file to your project.

Click Next to begin using this wizard.

Filter Files Panel

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Dynamic Scanning Wizard may list as dependencies certain files that you do not want to add to your installation. For example, common system files that are already present on target machines usually do not need to be reinstalled. To avoid having these files added to your project when you run the scanner, select the **Filter files** check box on the Filter Files panel.

To learn how to customize the list of files that are excluded from scans, see Filtering Files in Dependency Scanners.

Specify the Executable Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Specify the Executable panel is where you specify whether you want to scan an executable file that is already included in your project, or one that has not yet been added.

Table 12-42 • Specify the Executable Panel Settings

Setting	Description
I would like to select an executable file from my project (recommended)	If the executable file that you would like to scan has already been added to your project, select this option.
I would like to select a new executable	If the executable you want to scan is not currently included in your project, select this option.

Specify Application File Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Specify Application File panel is where you to select the specific executable file (.exe) that you would like to scan. Additionally, you can specify command-line parameters for the file and a working folder.

Table 12-43 • Specify Application File Panel Settings

Setting	Description
Application	Specify which executable file you would like to scan. If you choose to scan a file that is already included in your project, select it from the list of executable files that present in your project. If you choose to scan a file that is not yet a part of your project, enter the path to that file or click the Browse button to navigate to it.
Command Line	Enter the command-line parameters that you would like to pass to the executable file. These parameters are used only during the scanning process; they are not used after the wizard has been dismissed.
Working Folder	Enter the path to the working folder for this application, or click the Browse button to navigate to the directory. By default, this directory is set to the same folder in which the application you choose to scan resides.

Launch the Application



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Before the Dynamic Scanning Wizard begins scanning your application, it must launch the application. After your application is launched by the wizard, you should use as many of the application's menu items and features as you can. This helps to identify where dependency files are located so that they can be added to your project.

Click Next to launch your application and begin scanning for dependencies.

Your Application is Running Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Your Application is Running panel is displayed while your application is running. When you have finished using your application, click the Done button to view the results of the scan. No files are added until you have confirmed the findings of the scan.

File Selection Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The File Selection panel displays a list of possible files and merge modules that you may need to add to your project. Use this panel to select the ones that you want to include in your installation. For more information, see Reviewing Dependency Scanner Results.

Table 12-44 • File Selection Panel Settings

Setting	Description
File	Indicate which files and merge modules you want to add to your project by selecting the appropriate check boxes.
Deselect All	If you want to clear all of the check boxes, click this button. You can then manually select each check box that corresponds with a file or merge module that you would like to add to your project.
Select All	If you want to select all of the check boxes, click this button. You can then manually clear each check box that corresponds with a file or merge module that you do not want to add to your project.

Scan Results Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Scan Results panel shows the dependencies that the wizard identified and that you selected to be added to your project.

To add these dependencies to your project, click the Next button. To exit the wizard without adding the dependencies, click the Cancel button. To review the list of potential dependencies again and add or remove any of them, click the Back button.

Completing the Dynamic Scanning Wizard Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

When the Dynamic Scanning Wizard shows the Completing the Dynamic Scanning Wizard panel, the wizard has added the executable file's dependencies to your project. If you chose to scan an executable file that was not already included in your project, that .exe file has been added as well.

Click Finish to close the wizard and return to InstallShield.

Export Components Wizard



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Export Components Wizard simplifies the process of exporting a component from certain file types (.ism, .msi, .msm) to any existing project or a new one altogether. A common use of this wizard includes exporting components to merge modules.



***Note** • The Export Components Wizard does not let you export components from an .msi or .msm file to an .ism file. It also does not let you export from an .ism file to an .msi or .msm file.*

When you export components to other projects, a copy of this component is added to the specified project file, along with all of the component's data, such as its files, shortcuts, registry entries, and advanced settings.



Task ***To launch the Export Components Wizard:***

1. In the View List under **Organization**, click **Setup Design** (in installation projects only) or **Components**.
2. Do one of the following:
   - To export a specific component, right-click the component that you want to export, and then click **Export Component Wizard**.
   - To export more than one component, right-click the main node in this view, and then click **Export Component Wizard**.



***Note** • You can also launch the Export Components Wizard from the Project menu.*

The following panels are associated with the Export Components Wizard:

- Welcome
- Select Components
- Select Target File Info
- Summary

Welcome Panel



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Export Components Wizard simplifies the process of exporting components from a file type (.ism, .msi, .msm) to any existing project or a new one altogether. A common use of this wizard includes exporting components to merge modules.



Note *• The Export Components Wizard does not let you export components from an .msi or .msm file to an .ism file. It also does not let you export from an .ism file to an .msi or .msm file.*

Select Components Panel



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Select Components panel lists all of the components in your open project. Select the components that you want to export. You can filter the components list in this panel by feature or component destination.

If you launch the Export Components Wizard by right-clicking a specific component and then clicking Export Components Wizard, the wizard skips the Select Components panel.

***Note** • When you are exporting from a merge module, the feature filter is not available because there are no features in merge modules.*

Select Target File Panel

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

On the Select Target File panel, specify whether you want to export the component to an existing or new project. In either case, browse for the file location.

***Note** • When the wizard is exporting the component to the target project, the wizard checks for conflicts. If the target project already has a component of the same name with different settings, the Resolve Conflict dialog box opens to offer you options for resolving the conflicts.*

There is also an option to export as a Merge Module project. Select this option to export the component to this specific project type.

Target File Info Panel

***Project** • This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

On the Target File panel, specify information, including the product name, version, and language, for the new project. Indicate whether you want to replace exported components in the current project with the generated merge module. Replacing the component with the exported merge module option removes the component from the currently open project.

Summary Panel

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Summary panel displays the results of the export process. The summary also lists any errors from exporting. Click Finish to complete the process.

Note *• After you have arrived at the Summary panel, you cannot click Back to return to previous panels.*

Function Wizard

The Function Wizard automates the process of adding function calls to your scripts. After you complete two brief wizard panels, your function call is inserted into your script at the caret location.

Launching the Function Wizard

Task ***To launch the Function Wizard, do one of the following:***

- From the **Script Editor**, press Ctrl+I.
- Click the toolbar's **Insert InstallScript Function** button ( ).
- On the **Edit** menu, point to Insert and select **InstallScript Function**.

Function Name Panel

Lets you select an InstallScript function to paste into the active script editor window.

The Function Wizard lets you select from an alphabetic list of all functions or from lists of specific function categories. You can view the description for each function you highlight. Press the Help button to see the complete function description for any highlighted function.

Panel Options

Function Category (list box)

Select a function category. The InstallScript functions belonging to the selected category are displayed in the Function Name list box. Select "All" to display the names of all available InstallScript functions.

Function Name (list box)

Select a function name to display the function parameters and a brief description of the selected function in the bottom portion of this panel.

Click Next to display the Function Parameters panel of the Function Wizard.

Function Parameters Panel

Displays the name of the function selected in the Function Name panel and the parameters for that function. Next to each parameter is either an edit box in which you can enter a variable name or a value, or a drop-down menu from which you can select a parameter value.

Task ***To specify function parameters and insert the function into your script:***

1. Complete the arguments for each function. When you place the cursor in an edit field, the parameter description appears at the bottom of the panel. If there are predefined options available for a parameter, the edit field contains a drop-down menu. Note that you can select only one option from a drop-down menu, even when the options are not exclusive.

   Note • *You can enter strings (or string identifiers) and numbers directly in the edit fields, but you must pay careful attention to the purpose of the parameters. One clue is the InstallScript Hungarian notation in the variable name.*

   You must declare all variable names that you use in all function calls. The Function Wizard does not declare them for you.

2. Click **Finish** to insert the function call into your setup script. If you need further help, place the cursor in the function name and press the **F1** key.

Import DIM Wizard

Project • *This information applies to Basic MSI projects.*

Use the Import DIM Wizard to identify a DIM that you want to statically import into a Basic MSI project. When you complete the wizard, InstallShield incorporates the project data of the DIM into the open Basic MSI project.

Note that statically importing a DIM is a design-time task. If you statically import a DIM into a project, its data becomes part of the Basic MSI project at the time of the import. If you later update the DIM project, the Basic MSI project is not updated with the latest changes in the DIM. In order to remove a statically imported DIM from a Basic MSI project, it would be necessary to identify each file, component, and other element from the DIM and manually remove it from the Basic MSI project.

Task ***To launch the Import DIM Wizard:***

1. In the View List under **Organization**, click **Setup Design**.
2. In the **Setup Design** explorer, right-click the feature that you want to contain the DIM, and then click **Import DIM Wizard**.

The Import DIM Wizard consists of the following panels:

- Welcome
- Select Source File
- Import

Welcome Panel

Project • *This information applies to Basic MSI projects.*

Use the Import DIM Wizard to identify a DIM that you want to statically import into a Basic MSI project. When you complete the wizard, InstallShield incorporates the project data of the DIM into the open Basic MSI project.

Select Source File Panel

Project • *This information applies to Basic MSI projects.*

On the Select Source File panel, specify the path and file name for the DIM that you want to statically import into your Basic MSI project. To navigate to the file without having to manually enter the path and file name, click the Browse button.

If you want to back up your Basic MSI project file (.ism) before statically importing the DIM, specify the path and file name for the backup project file. If you specify a path and file name, the wizard saves the backup file when you click the Next button on this panel.

Import Panel

Project • *This information applies to Basic MSI projects.*

The Import panel shows the location and name of the DIM project file that you are importing. If you specified to create a backup copy of your Basic MSI project file (.ism) before statically importing the DIM, the path and file name are also displayed.

To complete the import process, click the Finish button.

Import REG File Wizard

InstallShield enables you to import existing registry (.reg) files that you have created in other installation projects or outside InstallShield.

Task ***To launch the Import REG File Wizard:***

1. In the View List under **System Configuration**, click **Registry**.
2. *Windows Installer projects only*: In the **View Filter** list, click the component that should have the .reg file that you are importing.

   Note *• If All Application Data is selected, the Registry view is read-only.InstallScript projects only: In the Destination computer's Registry view pane, open the registry set that should have the .reg file that you are importing.*
3. In the **Destination computer's Registry** view pane, right-click the registry key to which you would like your registry data added, and then click **Import REG File**. The Import Reg File Wizard opens.

When you import an .reg file into a component or registry set, that registry data is added to the component's or registry set's registry data and written to the end user's system if the component or registry set is installed.

The Import REG File Wizard consists of the following panels:

- Welcome
- Import Registry File
- Import Conflict Options
- Progress

Welcome Panel

The Import REG File Wizard allows you to import existing registry data (.reg) into your InstallShield project. This registry data is added to the target system's registry during the setup project.

Click Next to continue to the next wizard panel to begin importing your REG file.

Note *• Before you begin importing REG data be sure that you selected the proper feature to which you would like this data added. To specify a feature, cancel the wizard and select the appropriate feature from the Feature list at the top of the Registry view.*

Import Registry File Panel

In this panel you can specify the REG file you would like to import.

Panel Options

Registry File

Enter the path to the REG file you would like to import, or click the Browse button to navigate to this file.

Import Conflict Options Panel

Because your setup may already contain registry data that could conflict with information stored within the REG file you are importing, you can select how you would like to handle any conflicts.

Panel Options

Overwrite the registry data

Select this option if you would like data stored within the REG file you are importing to overwrite any conflicting data already present in your setup project.

Do not overwrite the registry data

Select this option if you would like to retain the data already present in your setup project when a conflict arises during the import progress. All nonconflicting registry data are still imported.

Log all registry conflicts to a file

Select this option if you want to keep a record of all registry conflicts found by the wizard. You can type a path and folder name in the field or click the folder button to browse to a file.

Import Progress Panel

This panel displays the import progress of the REG file. Click Cancel to stop the import or wait until the wizard finishes importing your REG file and click Finish to return to the IDE.

Import XML Settings Wizard

The Import XML Settings Wizard enables you to add a reference for an XML file to the XML File Changes view, where you can configure the changes that you want to be made to the XML file at run time.



Important • *The XML File Changes view is not designed to list a node for every node in an XML file. To improve performance, the XML File Changes view should show only the settings that differ from the base XML file:*

- *If the XML file that you are modifying is part of your installation, the XML File Changes view should list only the nodes and node sets that should be added, changed, or deleted after the XML file is installed at run time.*
- *If the XML file that you are modifying is a file that is already present on the target system, the XML File Changes view should list only the nodes that need to be added, changed, or deleted at run time.*

Therefore, when you use the Import XML Settings Wizard, import only the nodes in the XML file that you want to modify at run time.

Note that the XML File Changes view does not enable you to specify the order in which new elements should be listed in the XML file. Therefore, importing only the nodes that you want to modify at run time helps to avoid issues that may occur if your product requires that the elements be listed in a particular order.

The wizard consists of the following panels:

- Welcome
- XML File
- XML Element
- XML Import Progress
- XML Import Results

Welcome Panel

The Import XML Settings Wizard enables you to add a reference for an XML file to the XML File Changes view, where you can configure the changes that you want to be made to the XML file at run time.

The Welcome panel is the first panel of the Import XML Settings Wizard.



***Important** • The XML File Changes view is not designed to list a node for every node in an XML file. To improve performance, the XML File Changes view should show only the settings that differ from the base XML file:*

- *If the XML file that you are modifying is part of your installation, the XML File Changes view should list only the nodes and node sets that should be added, changed, or deleted after the XML file is installed at run time.*
- *If the XML file that you are modifying is a file that is already present on the target system, the XML File Changes view should list only the nodes that need to be added, changed, or deleted at run time.*

Therefore, when you use the Import XML Settings Wizard, import only the nodes in the XML file that you want to modify at run time.

Note that the XML File Changes view does not enable you to specify the order in which new elements should be listed in the XML file. Therefore, importing only the nodes that you want to modify at run time helps to avoid issues that may occur if your product requires that the elements be listed in a particular order.

XML File Panel

On the XML File panel, specify the XML file that you want to import by clicking Browse to locate it and populate the field. Then click Next to continue.

XML Element Panel

On the XML Element panel, specify which elements in the XML file should be imported into the XML File Changes view. Then click Import to begin the process or click Back to return to the previous panel.



***Important** • The XML File Changes view is not designed to list a node for every node in an XML file. To improve performance, the XML File Changes view should show only the settings that differ from the base XML file:*

- *If the XML file that you are modifying is part of your installation, the XML File Changes view should list only the nodes and node sets that should be added, changed, or deleted after the XML file is installed at run time.*
- *If the XML file that you are modifying is a file that is already present on the target system, the XML File Changes view should list only the nodes that need to be added, changed, or deleted at run time.*

Therefore, on the XML Element panel of the Import XML Settings Wizard, select only the nodes in the XML file that you want to modify at run time.

Note that the XML File Changes view does not enable you to specify the order in which new elements should be listed in the XML file. Therefore, importing only the nodes that you want to modify at run time helps to avoid issues that may occur if your product requires that the elements be listed in a particular order.

XML Progress Panel

The XML Progress panel appears after you click Import in the Import XML Element panel. It displays the progress of the file import.

XML Results Panel

The XML Results panel displays a summary of the imported .xml file and all of its elements. Click Finish to complete the import process and exit the wizard.

Once you click Finish, InstallShield adds a new node for the XML file that you imported. The XML file node contains each of the nodes that you selected in the wizard. Each node represents an XPath query that occurs at run time. InstallShield also adds a new component for the XML file that you have imported through the XML File Changes view. Once you are done importing, you can use the XML File Changes view to configure the XML file's settings and specify any element, attribute, and content changes for it.

For examples of how to create some basic XPath expressions, see Using XPath Expressions to Find XML Data in an XML File.

Microsoft .NET Framework Object Wizard

Project *• The Microsoft .NET Framework Object Wizard is available in InstallScript projects.*

The Microsoft .NET Framework Object Wizard opens when you add the Microsoft .NET Framework Object to a feature in the Objects view of an InstallScript project. This wizard lets you specify which versions of the .NET Framework and language packs should be included in your project. The language packs contain translated text, such as error messages, for languages other than English.

The Microsoft .NET Framework Object Wizard consists of the following panels:

- Step 1
- Step 2

Tip *• This object is not installed with InstallShield; you need to download it. For more information, see Obtaining Updates for InstallShield.*

Step 1 Panel



Project • *The Microsoft .NET Framework Object Wizard is available in InstallScript projects.*

The Step 1 panel of the Microsoft .NET Framework Object Wizard lets you specify which versions of the .NET Framework and language packs should be included in your project.

Table 12-45 • Step 1 Panel Settings

Setting	Description
Select the version of .NET to include and install	Select which versions of the .NET Framework should be included in the object.
Use Web downloader version of .NET redistributable	If a Web downloader redistributable and a full package redistributable are available for the selected version of .NET, this check box is enabled. Select this check box if you want to use the Web downloader redistributable. The Web downloader redistributable is smaller than the full package redistributable, but it requires that the target system have an Internet connection at run time.
Platforms	Select which platform-specific versions of the .NET Framework should be included in the object.
Languages	Specify whether the object should include language packs. • **Don't include any language packs**—If no language packs should be included, select this option. • **Include all appropriate language packs**—If the object should include all of the available language packs that are supported by the project, select this option.

For more information about this object, see the help pane that is displayed when you click the Microsoft .NET Framework Object in the Objects view.



Tip • *This object is not installed with InstallShield; you need to download it. For more information, see Obtaining Updates for InstallShield.*

Step 2 Panel



Project • *The Microsoft .NET Framework Object Wizard is available in InstallScript projects.*

If you select the **Include multiple versions of the .NET Framework** option on the Step 1 panel of the Microsoft .NET Framework Object Wizard, the Step 2 panel is displayed.

Table 12-46 • Step 2 Panel Settings

Setting	Description
Select the .NET redists to include in the object	Select the check box for each version of the .NET Framework that should be included in the project. Note that the object can install only one version of the .NET Framework at run time.
Allow the object to install a .NET language pack without installing the corresponding .NET Framework.	To allow a language pack to be installed without first installing the corresponding version of the .NET Framework, select this check box. InstallShield uses this setting to configure the **InstallLangPackOnly** property for the object. This read-write numeric property indicates whether the object should allow the installation of the language pack without first installing the corresponding version of the .NET Framework, which may occur if the framework is not being included in the object. By default, if the object is set to install a particular version of the framework and that version of the framework is not included in the object, the object sets a status of DOTNET_STATUS_VERSION_NOT_IN_MEDIA. However, if the **InstallLangPackOnly** property is set, the object does not set the failure status, which allows the object to install the language pack, if appropriate, without first installing the framework. This property is typically set in the object wizard.

For more information about this object, see the help pane that is displayed when you click the Microsoft .NET Framework Object in the Objects view.

Tip • *This object is not installed with InstallShield; you need to download it. For more information, see Obtaining Updates for InstallShield.*

New Language Wizard

Edition • *The New Language Wizard is available in the Premier edition of InstallShield.*

Project • *The New Language Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

The InstallShield Premier edition lets you add unsupported languages, beyond the built-in 35 languages, to projects through the New Language Wizard. An unsupported language is one in which none of the default run-time strings are translated. When you add an unsupported language to a project, that language is made available in various language-related settings throughout InstallShield. In addition, InstallShield uses the strings from your project's default language as placeholders for the strings in that newly added unsupported language; you can use the String Editor view to provide translated strings for the unsupported languages.

When you use the New Language Wizard to add an unsupported language, InstallShield adds to your machine several files that contain English strings as placeholders:

- *LanguageID*.ini

  LanguageID is the language identifier for the new language. InstallShield adds this file to the following directory:

  *InstallShield Program Files Folder***Support**

- _isres_*LanguageID*.dll

  LanguageID is the language identifier for the new language. InstallShield adds this file to the following directories:

  *InstallShield Program Files Folder***Redist\Language Independent\i386**

  *InstallShield Program Files Folder***Redist\Language Independent\i386\Skins**

If you want to add the unsupported language in InstallScript or InstallScript MSI projects, you will need to translate the strings in those files. You can open the .dll file in a resource editor to replace the English strings with the appropriate translated strings.

Task ***To launch the New Language Wizard:***

On the **Tools** menu, click **Add New Language**.

Welcome Panel

Edition • *The New Language Wizard is available in the Premier edition of InstallShield.*



Project *• The New Language Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

The InstallShield Premier edition lets you add unsupported languages, beyond the built-in 35 languages, to projects through the New Language Wizard. An unsupported language is one in which none of the default run-time strings are translated. When you add an unsupported language to a project, that language is made available in various language-related settings throughout InstallShield. In addition, InstallShield uses the strings from your project's default language as placeholders for the strings in that newly added unsupported language; you can use the String Editor view to provide translated strings for the unsupported languages.

Project Language Panel



Edition *• The New Language Wizard is available in the Premier edition of InstallShield.*



Project *• The New Language Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

The Project Language panel lets you specify the languages that you would like to add to your project. Select the box next to each language that you want to add.

Project Files Panel



Edition *• The New Language Wizard is available in the Premier edition of InstallShield.*



Project *• The New Language Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

The Project Files panel is where you specify which existing projects should contain the ability to add the new languages. (That is, InstallShield will add the new languages to the list of languages that you can select for the Setup Languages setting in the General Information view.)

This panel also enables you to specify whether you want to be able to include the languages in every new project that you create.

For information on adding a language to a project, see Selecting the Installation Languages.

Panel Settings

Table 12-1 • Project Files Panel Settings

Setting	Description
Project Name	Select the projects that should enable you to select the new languages for the Setup Languages setting in the General Information view. If your project is not stored in the default project location, navigate to it by clicking the Browse button.
Update InstallShield	Select this check box if you would like any new project that you create to have optional support for these languages. Selecting this check box does not automatically include the new languages in future projects. Instead, it adds the new languages to the list of languages that you can select for the Setup Languages setting in the General Information view.

Summary Panel



Edition • *The New Language Wizard is available in the Premier edition of InstallShield.*



Project • *The New Language Wizard is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Suite/Advanced UI*

The Summary panel lists the new languages that will be added to the specified projects. It also indicates whether you want InstallShield to add the new languages to all new projects that you create. Click the Finish button to accept the settings and begin updating projects with the new languages.

When you use the New Language Wizard to add an unsupported language, InstallShield adds to your machine several files that contain English strings as placeholders:

- *LanguageID*.ini

  LanguageID is the language identifier for the new language. InstallShield adds this file to the following directory:

  *InstallShield Program Files Folder***Support**

- _isres_*LanguageID*.dll

  LanguageID is the language identifier for the new language. InstallShield adds this file to the following directories:

*InstallShield Program Files Folder***Redist\Language Independent\i386**

*InstallShield Program Files Folder***Redist\Language Independent\i386\Skins**

If you want to add the unsupported language in InstallScript or InstallScript MSI projects, you will need to translate the strings in those files. You can open the .dll file in a resource editor to replace the English strings with the appropriate translated strings.

Open MSI/MSM Wizard

The Open MSI/MSM Wizard is a tool that converts existing installation projects (.msi files) and merge modules (.msm files) to InstallShield installation projects (.ism files) that you can modify and build in the InstallShield user interface.



Task ***To launch the Open MSI/MSM Wizard:***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. In the file type list, select either **Windows Installer Packages (*.msi)** or **Windows Installer Modules (*.msm)**.
3. Browse to the Windows Installer package or merge module that you want to convert.
4. On the **Open** button, click the arrow, and then click **Open with Wizard**.

The Open MSI/MSM Wizard opens.



Note • *If the wizard encounters errors while opening the package, the errors are displayed in the Output window when the converted project is opened. To solve any problems, see MSI/MSM Conversion Errors.*

Welcome Panel

The Open MSI/MSM Wizard allows you to import setup packages (.msi) and merge modules (.msm) into InstallShield. This functionality is useful for network administrators who need to build custom setups for their users.

Panel Options

Open MSI/MSM in Direct Edit Mode

Select this option to view and edit the database directly, without creating a project file. If you select this option, the selected database will open in the IDE, where you can view and edit its contents with the Direct Editor.

Click Next to open the database in the Direct Editor.

Convert MSI/MSM to an InstallShield Project

Select this option to convert the selected database into an InstallShield project, which you can then edit within the IDE.

Click Next to continue using the wizard for this option.

File Locations Panel

In the File Locations panel, specify the name of the project that the wizard will create and the location of its data files.

Panel Options

Project Name

Type a name for the setup project that will be created when the wizard converts the existing .msi or .msm project to an InstallShield (.ism) project. The .ism project is saved in the default location.



Note • *Because the project name determines the name of the .ism file and the root for the project's folder structure, use only legal characters for a Windows file name.*

Data File Location

If the imported application's files are stored in .cab files, those files are automatically extracted and stored in the location specified here.

After specifying the project name and data file location, click Finish to convert the project and open it in the InstallShield IDE.



Note • *The wizard alerts you of any errors it encountered while creating the project. To resolve any problems, see MSI/MSM Conversion Errors.*

Open MSP Wizard

Before InstallShield can open a patch or MSP file for editing or create a new patch project, it needs the base MSI to which the MSP should be applied. When you first open a patch (.msp) project in InstallShield, the Open MSP Wizard appears.

The Open MSP Wizard gathers the necessary file information, applies the patch to the base MSI, and opens the patch project in the IDE in Direct MSP mode (which is similar to the Direct Edit mode). The data you specify in the Open MSP Wizard is saved so the next time you open the MSP file, the patch project opens in the IDE.

Welcome Panel

This panel welcomes you to the wizard and briefly describes the Open MSP Wizard's function. Click Next to begin using the wizard.

Base MSI Package Panel

In the Base MSI Package panel, you specify the base MSI file to which you want to apply this patch.

Panel Options

MSP File Name

Read-only field that provides the path and file name of the MSP file.

Base MSI File Name

Specify the name of the base MSI file to which this patch (indicated in the MSP File Name edit box) will be applied. Click the browse (ellipsis) button to browse to an .msi file.

Click Finish to close the Open MSP Wizard and create your patch project. The patch is applied to the MSI project that is specified in the Base MSI File Name edit box, and the patch project (.msp file) opens in Direct MSP mode in the IDE.

Open Transform Wizard

Before InstallShield can open a transform (.mst) file for editing or create a new Transform project, it needs the base .msi file to which the .mst should be applied and any additional .mst files that should be applied to the base .msi file.

The first time that you open an .mst file in InstallShield, the Open Transform Wizard launches. This wizard enables you to specify the .mst file, the base .msi file, and any additional transforms to apply before editing. The last panel of this wizard—the Create a Response Transform panel—lets you specify whether or not your transform should be a response transform. A response transform includes default responses for the user interface portion of the installation. If you specify that you want to create a response transform, the user interface elements of the installation are executed, enabling you to customize the options available in each panel of the installation wizard.

When the Open Transform Wizard is finished, the transforms are applied and the .mst file opens in the InstallShield in Direct MST mode (which is similar to Direct Edit mode). The data specified in the wizard is saved so that the next time that you open the .mst file, the transform project opens directly in InstallShield.

Task ***To launch the Open Transform Wizard when opening a transform for editing:***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. Select the .mst file that you would like to open.
3. In the **Open as** box, select **Wizard**.
4. Click **Open**. The Open Transform Wizard opens.

Task ***To launch the Open Transform Wizard when creating a new transform:***

1. On the **File** menu, click **New**. The **New Project** dialog box opens.
2. On the **Windows Installer** tab, click **Transform**.
3. In the **Project Name** box, type a name for your transform project.
4. In the **Location** box, type the path where your transform project should be stored or click the browse button to find the location.

5. Click **OK**. The Open Transform Wizard opens.

The following panels are associated with the Open Transform Wizard:

- Welcome
- Transform Information
- Additional Transforms
- Create a Response Transform

Welcome Panel

This panel welcomes you to the Open Transform Wizard and briefly describes the wizard's function. Click Next to begin using the wizard.

Transform Information Panel

In the Transform Information panel, you specify the base .msi file to which you want to apply this transform.

Panel Options

MST File Name

Read-only field that provides the path and file name of the .mst file.

Base MSI File Name

Specify the name of the base .msi file to which this transform (indicated in the MST File Name box) will be applied. Click the ellipsis (...) button to browse to an .msi file.

Click Next to move to the next panel.

Additional Transforms Panel

In this panel, you can supply the names of one or more additional transforms that you want applied prior to applying the transform that the Open Transform Wizard is currently creating. For example, you may want to include previous customization or transforms containing language-specific information. The transforms are applied in the order in which they are listed.

Adding a Transform

Task ***To add a transform to the list:***

1. Click New (Insert) or press Insert.
2. Type the full path and name of the .mst file or click the ellipsis (...) button to browse to it.

Deleting a Transform from the List

Task ***To delete a transform from the list:***

1. Select the transform in the list that you would like to remove.
2. Click Delete or press Delete.

Changing the Transform Order

Task ***To change the order in which the transforms are applied:***

Select a transform in the list and click the move up button or the move down button.

The transform that the Open Transform Wizard is creating is applied last.

Caution • *When you are using multiple transforms, keep in mind that the order in which they are applied is critical. For example, if you create a transform for a Windows Installer package that creates a new value for a property, and then you create a second transform that changes the value created in the first transform, everything works correctly. However, if you apply the second transform first, that transform is attempting to modify the property's value, instead of creating it. That will result in an error.*

One simple example of where this may be a problem is with the default company name. If the value is not set by default, and you set it in the first transform, a new value for the property is created. If you create a second transform that modifies the combined original package and first transform, and the second transform changes the default company name, it is only changing the property. However, if you try to apply the second transform without the first one, Windows Installer interprets this as trying to change a null value to another value, which will result in an error.

Click Next to move to the next panel.

Create a Response Transform Panel

In this panel, you can specify that you want to create a response transform.

Panel Options

Table 12-2 • Create a Response Transform Panel Options

Option	Description
Create response transform	If you want to create a response transform, select this check box. If you select this option, the user interface elements of the installation are executed when you click Finish. No file transfer takes place, and no changes to your machine occur. Complete this simulated installation, customizing the options available in each panel of the installation wizard as needed. When you reach the end of the installation sequence and click Install, the simulated installation exits. InstallShield saves all of the changes that you made during the simulated installation as default installation values in the response transform.
Command line properties (optional)	If you are using a response transform, you can specify additional command-line properties (in property name/value pairs separated by semicolons) to pass to the response transform. These must be public properties, and they control only how the dialog boxes are displayed during creation of the response transform. They are not persisted outside of the user interface sequence during creation. For example, you can pass the property/value pair ARPHELPTELEPHONE=1-111-111-1111 to set the value of the Help Telephone field of Add or Remove Programs in Control Panel. You might pass a property/value pair during the creation of a response transform to display all dialog boxes during an installation that may not be displayed based on your system configuration (for example, to show Windows 9x–only dialog boxes on a Windows NT platform). You can then make appropriate responses and have them included in your transform.

Click Finish to exit the Open Transform Wizard and create your transform project. The transforms are applied to the .msi file specified in the Transform Information panel, and the transform project (.mst file) opens in Direct MST mode in InstallShield.

Publish Wizard

Use the Publish Wizard to save the selected installation element to a repository.

The following panels are associated with the Publish Wizard:

- Welcome
- Repository
- Publishing Information

- Summary

Welcome Panel

This panel welcomes you to the Publish Wizard. Click Next to begin using the wizard.

Repository Panel

The Repository panel of the Publish Wizard is where you specify the repository to which your installation element should be published.

Publishing Information Panel

The Publishing Information panel of the Publish Wizard is where you specify information that identifies the item that you are publishing to the repository. If you are publishing a dialog to the repository, for example, the information that you provide in the Publishing Information panel will be displayed for your dialog when you or other users browse the repository of dialogs.

Panel Options

Name

Specify a name for the dialog, template, script file, or other installation element that you are publishing.

Publisher

Type your name.

Description

Type the description that should be displayed for this installation element.

Summary Panel

Use the Summary panel of the Publish Wizard to review the settings for the item that you are publishing to the repository. If you need to change anything, click Back until you reach the appropriate panel, and then make the necessary changes.

When you are done, click Finish. The Output window opens to inform you whether the installation element was successfully published to the selected repository.

Redistributable Downloader Wizard



Project *• This information applies to the following project types:*

- *Basic MSI*

- *InstallScript MSI*

The Redistributable Downloader Wizard allows you to quickly download third-party redistributables, merge modules, and other files to your local machine. You can launch the Redistributable Downloader Wizard from the Release Wizard's .NET Run-Time Options panel and from the Releases view (.NET 1.1/2.0 Core Language and .NET 1.1/2.0 Language Packs settings).



Task ***To use the Redistributable Downloader Wizard:***

1. In the **Select Files to Download** panel, select the redistributables that you want to download and click **Next** to move to the **Progress** panel.
2. When the progress bars complete, click **Next**.
3. The final panel displays the list of redistributables that the wizard downloaded to your machine. Click **Finish** to exit the wizard.

Reg-Free COM Wizard

Use the Reg-Free COM Wizard to create and modify Reg-Free COM manifest files to be included in your installation.



Note *• Before you use the Reg-Free COM Wizard, you should add the COM libraries (.dll and .ocx files) and the executable file that uses them to your InstallShield project. Note that the Reg-Free COM manifest file, the executable file, and the COM libraries should all be installed to the same folder on the target machine.*



Task ***To launch the Reg-Free COM Wizard:***

On the **Project** menu, click **Reg-Free COM Wizard**.

The following panels are associated with the Reg-Free COM Wizard:

- Welcome
- Select an EXE File
- Select the Manifest File
- Select Files
- Summary

Welcome Panel

The Welcome panel provides a brief introduction to the Reg-Free COM Wizard. Click Next to begin using the wizard.



Note *• Select the* ***Don't show the welcome panel again*** *check box if you do not want this panel to appear the next time you open the wizard.*

Select an EXE File Panel

The Select an EXE File panel is where you specify the executable file that uses the manifest file that you are creating or modifying.

Panel Options

Executable FileKey

Select an executable file from the Executable FileKey list and click Next. Note that the file must already be included in your installation project.

Select the Manifest File Panel

The Select the Manifest File panel is where you either specify that you are creating a new manifest (.manifest) file or that you are modifying an existing one. If you are modifying a manifest file, you need to specify where it is located.

Panel Options

Create a new manifest file that will be added to the project

To create a new manifest file, select this option.

Pre-existing file that will be added to the project

To specify a manifest file that already exists but has not been included in the project yet, select this option, and then click the Browse button to locate the file.

A manifest file that already exists in the project

To specify a manifest file that has already been added to the project, select this option and then select the file from the list.

Select Files Panel

The Select Files panel is where you specify the required COM library files (.dll or .ocx) that are used by the executable file. You can additionally specify to extract the COM information during each build by selecting the **Refresh COM Info during Build** check box. If you clear this check box, the COM information is injected into the manifest only once, when you have completed the wizard.

Summary Panel

Use the Summary panel of the Reg-Free COM Wizard to review the options that you selected. If you need to change anything, click Back until you reach the appropriate panel, and then make the necessary changes.

When you are done, click Finish. The wizard creates a new component with the manifest file set as the key file. The manifest component has the same destination and condition as the associated executable-file component. The following file naming convention is used for manifest files:

ExecutableFileName.exe.manifest

For example, **MyFile.exe**.manifest is created for an executable file called **MyFile.exe**.

You can specify additional COM servers by running the wizard again and specifying the existing manifest file.

Release Wizard

The Release Wizard provides an easy way for you to build a release for your product and specify the settings particular to that release.

Task ***To launch the Release Wizard and build your setup package, do one of the following:***

- Click the **Release Wizard** button on the toolbar.
- Select **Release Wizard** from the **Build** menu.
- Double-click the **Release Wizard** item in the **Release** view.
- Right-click on a product name or a release name in the tree control in the **Release** view, and select **Release Wizard**.

Task ***You can launch an abbreviated version of the Release Wizard by doing one of the following:***

- Click the **Build** button on the toolbar.
- Select **Build** from the **Build** menu.
- Right-click on a release name in the Release explorer, and select **Build**.

The Build option rebuilds the release that has focus in the Release view or builds the product's first release with default settings. The build feedback—and any build errors—appears in the Output window.

Another alternative to the Release Wizard is building a release at the command line.

Welcome Panel

This panel provides a brief introduction to the Release Wizard. This wizard walks you through all the necessary steps required to build a distributable release of your product.

Click Next to begin using the wizard.

Product Configuration Panel

Project • *This topic does not apply to InstallScript projects.*

A product configuration is the highest level of organization in the Release view. By grouping related releases together under a common product configuration, you can change specific settings for each release by editing the properties of the product configuration.

Panel Options

New product configuration

Select this option if you want to build this release under a new product configuration. Specify a name for the new configuration.

Existing product configuration

Select this option to build this release under an existing product configuration. Choosing this option overwrites any existing release.

Specify a Release Panel

This panel gives you the option of creating a new release or rebuilding an existing release.

Panel Options

New Release Name

If you would like to build a new release, select the New Release Name option and then enter a new name in the field.

Existing Release Name

If you choose to build an existing release, click the Existing Release Name option and then select a release name from the drop-down list.

Filtering Settings Panel



Project • *The Filtering Settings panel is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

The Filtering Settings panel prompts you to select which features, components, InstallShield prerequisites, and chained .msi packages to include in your release.

Table 12-3 • Filtering Settings Panel Options

Setting	Description
Release Flags	To filter features, InstallShield prerequisites, and chained .msi packages, enter the release flags that you would like to include in this release. If you are including more than one flag, separate them with commas. You have the option of including a subfeature but not its parent feature. In such a case, the subfeature is built into the release as a top-level feature, and its parent is excluded from the release. The release flags that you enter here are combined with the product configuration flags. For more information on the relationship between the two, see Product Configuration Flags vs. Release Flags.  ***Project*** • *The Release Flags option is not available for merge module projects.*
Filter application data based on the following languages	To filter application data based on each component's language, select this option, and then select or clear the appropriate language check boxes. If you select one or more language check boxes, InstallShield includes in the release all of the components that are marked with a matching language, as well as all of the components that are marked as language independent; InstallShield excludes any components whose languages do not match.

Merge Module Options Panel

The Merge Module Options panel provides the option to make your new module immediately available to other InstallShield packages by copying it to your local merge module gallery.

Panel Options

Add to local merge module catalog

Select this option if you would like your new module to be immediately available to other projects. The Merge Modules Locations field is located on the Merge Modules tab of the Options dialog.

Setup Languages Panel



Project • *The functionality for the Setup Languages panel varies, depending on which project type you are using.*

The Setup Languages panel lets you specify language-related settings for your installation's run time.

Table 12-4 • Setup Languages Panel Settings

Setting	Description
Included Release Languages	The functionality for this box varies, depending on which project type you are using: • In a Basic MSI or InstallScript MSI project, this setting lets you specify which user interface languages you want to include in a release. Note that if a language is not selected in the Setup Languages setting in the General Information view of the project, it is not listed as one of the available languages in this box. • In an InstallScript or InstallScript Object project, use this setting if you want to include certain components and exclude others based on the language that is selected for each component. This setting also lets you specify which user interface languages you want to include in a release. If the language specified for a component does not match one of the languages that is selected in this box, InstallShield does not include the component in the release. In addition, if a UI language that is included in the project does not match one of the languages that is selected in this box, InstallShield does not include the UI strings in the release. By default, releases are language independent; that is, none of the project's components or UI strings are excluded from the release. Note that if a language is not selected in the General Information view of a project, it is not listed as one of the available languages in this box.
Make Default	To override the release's default project language that is configured in the General Information view or the String Editor view, select the appropriate default user interface language for your installation, and then click this button. For more information, see Setting the Default Project Language.
Display the Setup Languages Dialog	If you want your installation to display the Choose Setup Languages dialog when it is run with a full user interface, select this check box.  ***Tip*** • *A* **Setup.exe** *setup launcher is required if you want the language selection dialog to be displayed. To learn more, see Creating a Setup Launcher.*



Project *• In Basic MSI and InstallScript MSI projects, the Setup Languages panel affects only the user interface elements that you include in your project. To include components based on the language of the application data, see the Release Filtering panel.*

For information on adding a language to a project, see Selecting the Installation Languages.

Media Type Panel

This panel prompts you to select the type of distribution media for which you want to create a release.

Panel Options

Media type

Choose the type of media on which you will be distributing your setup program. You can select from the following options:

Table 12-5 • Media Type Panel Options

Option	Description
CD-ROM	Select this option if you will be distributing your setup on a CD. The maximum size for this type of media is 650 MB.
Custom	Select this option if the media that you are distributing is not in the list of media types. For example, you may be distributing your setup on floppy disks or Zip disks. You will need to set the format size and cluster size for this media, as it will change depending on the media type you are using.
DVD-5	This option allows you to ship your release on a 4.38 GB DVD-ROM.
DVD-9	Choose this option if you plan on shipping your setup on a 7.95 GB DVD-ROM.
DVD-10	This option allows you to ship your release on an 8.75 GB DVD-ROM.
DVD-18	If you have an enormous setup, you can ship your release on 15.83 GB DVD-ROM media.
Network image	If your setup is going to be placed on a network, you can choose this option. There is no size limit for this media type.
Web	(Windows Installer based projects only) If you want to distribute your release over the Web, choose this option. There is no size limit for this media type.



Project *• If your installation requires more than one disk, InstallShield automatically splits it across as many disks as needed. In a Windows Installer–based project, the wizard prompts you for more information about disk spanning later in the sequence.*

Format size

If you chose the Custom media type, you will need to specify the capacity of the media in this field. For example, if you are shipping your installation program on a Zip disk, you will want to set the media size to 100 MB. If you chose any other media type, the maximum size allowed for that media is displayed in this field.

Cluster size in bytes (Windows Installer based projects only)

This option defines the minimum amount of space that each file requires. For bigger disks, the cluster size will be larger due to the formatting constraints.

Disk Spanning Options Panel



Project • *This panel is not available for InstallScript projects.*

If your setup is larger than one disk, you have the option to split the setup into pieces so it can be distributed on multiple disks.

Panel Options

Automatic

Select this option to have the wizard automatically split your setup across as many disks as are necessary. This option is the recommended way to span your setup across disks.

Custom

Select this option to manually define how many disks your setup requires, and set the breakpoints yourself. This option requires you to specify which features will be stored on each disk in the Disk Spanning Advanced Settings panel.



Caution • *Multi-disk setups cannot be run from a non-removable media (for example, from a hard drive). If you want to run test a setup that spans disks, you need to put the setup on your target media. If you do not, the setup will fail due to a limitation of the Windows Installer service.*

Enforce disk size

Select this option if you want the wizard to verify that none of your disk images exceeds the media size that you selected. If any of the disks exceeds the specified size limit, Build Error -1531 appears when you try to build the project.

Disk Spanning Advanced Settings Panel



Project • *This panel is not available for InstallScript projects.*

This panel prompts you to specify the number of disks that your setup requires. In addition, you can choose which features are on each disk. This panel is displayed only if you selected the Custom Disk Spanning option on the Disk Spanning Options panel.

Panel Options

Disk Mapping

This dialog lists all the features included in the release when it was first built. Because a release is designed to be a snapshot of your build settings, features created after a release will not be available for disk mapping nor built into the current release. You must start a new release in order to map the new features.

All features are on Disk 1 by default. To change the name of this disk, click the Rename Disk button and type in the new name of the disk. To add a new disk, click the New Disk button.

Task ***To transfer some of your features from one disk to another:***

1. Select the feature that you want to move.
2. Click the **Move Up** or **Move Down** button. The feature will move through the hierarchy of the current disk until there is nowhere left to go, at which time it will skip to the next disk, provided there is another disk for it to move to.

Tip *• Note the following when spanning your setup across multiple disks:*

- *A single feature cannot be spanned across multiple disks. If you have a feature that is too big to fit on one disk, you need to divide it into smaller features or subfeatures.*
- *Any merge modules that are included in your setup must be on the first disk, because these files must be available when Windows Installer first processes the database (.msi file).*

If you add features to your setup after you have defined how you would like your setup to span across multiple disks, those features are automatically added to the last disk of your setup. You can add new disks to your setup at any time by using the Release Wizard.

Disk Prompt

Task ***To indicate the prompt you want displayed to end users when the next disk needs to be inserted:***

1. Select a disk in the **Disk Mapping** section.
2. Do one of the following in the **Disk Prompt** field.
   - Type the prompt that you want to display to end users.
   - Click the ellipsis button (**...**) to display the Select String dialog and select a string to use as the prompt.

Repeat for each disk except the last. The Release Wizard automatically creates a string-table entry for the Disk Prompt value, so you can provide a translated string for each language your setup program supports.

The string that is displayed comes from a number of sources throughout your project, as described below:

1. The complete prompt originates in the Error table (exposed in the Direct Editor) as Error record 1302.
2. This value in turn comes from your project's string entries, and the default value for English reads "Please insert the disk: [2]."
3. Windows Installer resolves *[2]* with the value of the property DiskPrompt, which is set to *[1]* in the Property Manager.
4. Finally, Windows Installer evaluates *[1]* to the string you enter into the Disk Prompt field.

In most cases you should enter the same name as the disk volume (below). For example, assuming the defaults are unchanged, a Disk Prompt entry of *Disk8* would cause the user to be prompted with "Please enter the disk: Disk8."

Disk Volume

Select a disk and then enter the name for the resulting disk image folder. Repeat for each disk.

When you are building your release image for removable media (such as CD-ROM), be sure to create the media with the same volume names specified here.

If you leave the names as DISK?, the Release Wizard numbers the disks sequentially (for example, DISK1, DISK2, and so on).



Tip *• Notice that the default name of the volume is in all uppercase letters. It is a good idea to keep the names as uppercase to accommodate media types, such as CD-ROM, which require volume names in that format.*

Web Type Panel



Project *• This panel is not available for InstallScript projects.*

The Web Type panel prompts you to select the configuration of your Web installation package.

Panel Options

One Executable

Select this option if you want to build this release as a single self-extracting **Setup.exe**. This Web type is ideal for a package that is to be downloaded from multiple Web or FTP sites, since the installation package is self-contained, and it does not contain any configuration information that ties it to a specific location.

Note, however, that the entire installation package will be downloaded, so the download size and time will be greater than the Install from the Web type described below.

Downloader

Select this option to build this release as a combination of **Setup.exe** and your .msi database. With this option, you can specify where your product's data files should be stored: they can be streamed into the .msi database or stored externally in cabinet (.cab) files. Any .mst and .ini files are embedded in the **Setup.exe** file.

With the Downloader option, the end user downloads and runs **Setup.exe**, which in turn downloads and runs the .msi database. If the .cab files are stored externally, only the .cab files that are required, based on the end user's setup type and feature selections, are downloaded and installed. This minimizes download size and time.

The URL that you specify in the Downloader Options panel of the Release Wizard is used during product maintenance and repair mode.

Install from the Web

Select this option to build this release as a combination of **Setup.exe**, your .msi database, and external cabinet (.cab) files. Any .mst and .ini files are embedded in the **Setup.exe** file.

With the Install from the Web option, the end user downloads and runs **Setup.exe**, which in turn downloads and runs the .msi database; based on the end user's setup type and feature selections, only the requested .cab files are downloaded and installed, which minimizes download size and time.

The URL that you specify in the Install from the Web Options panel of the Release Wizard is used during product maintenance and repair mode.

Downloader Options Panel



Project • *This panel is not available for InstallScript projects.*

The Downloader Options panel is where you specify the URL for your installation package. It is also where you specify the location for your product's data files.

Table 12-6 • Downloader Options Panel Settings

Setting	Description
URL for your package	Type the URL for your installation package—for example, **http://www.yourcompany.com/downloads**. (The file name is not necessary.)
Create external .cab files	To stream your product's data files into the .msi package, clear this check box. To store your product's data files outside the .msi package (in external .cab files), select this check box and then select the appropriate **.cab Files** option.

Table 12-6 • Downloader Options Panel Settings (cont.)

Setting	Description
.cab Files	If you select the **Create external .cab files** check box, specify how the (.cab files should be created. Available options are: • **One .cab per component**—Builds a separate cabinet file for each component. At run time, only the cabinet files that are required by the end user's setup type and feature selections are downloaded. • **One .cab per feature**—Builds a separate cabinet file for each feature. At run time, only the cabinet files that are required by the end user's setup type and feature selections are downloaded. • **Multiple .cab files based on size**—Enter a specific size, in kilobytes, for each cabinet file.

Install From The Web Options Panel

Project • *This panel is not available for InstallScript projects.*

The Install From The Web Options panel is where you specify the URL for your installation package. It is also where you specify how the .cab files should be created.

Table 12-7 • Install From The Web Options Panel Settings

Setting	Description
URL to install files from the Web	Type the URL for your installation package—for example, **http://www.yourcompany.com/downloads**. The URL must begin with **http://** or **ftp://**.
.cab Files	Specify how the (.cab files should be created. Available options are: • **One .cab per component**—Builds a separate cabinet file for each component. At run time, only the cabinet files that are required by the end user's setup type and feature selections are downloaded. • **One .cab per feature**—Builds a separate cabinet file for each feature. At run time, only the cabinet files that are required by the end user's setup type and feature selections are downloaded. • **Multiple .cab files based on size**—Enter a specific size, in kilobytes, for each cabinet file.

One-Click Install Panel

Project • *This panel is not available for InstallScript projects.*

The One-Click Install panel is where you specify whether to generate an HTML page and cabinet file from which an end user can begin installing your project.

Panel Options

Generate a One-Click Install

Select this option if you would like to create a One-Click Install, which is an installation program whose initial user interface is an HTML page. When an end user visits your Web page and clicks a button on it, the installation files are downloaded to the target system and then launched. The files that are downloaded to the user's system depends on your setting for the Web Type property for the current release.

HTML base file name

Type in the base name of the HTML file to generate, as in *install*. The ".htm" extension will be added to the base name you specify.

CAB base file name

Type in the base name of the cabinet (.cab) file to generate, as in **install**.

Local Machine Panel



Project • *This panel is not available for InstallScript projects.*

The Local Machine panel prompts you to specify whether and where your installation files should be cached on the target system.

Panel Options

Cache installation on local machine

Check this box if you would like your installation files to be cached on the target system. When the user performs maintenance operations on your product, the cached location you specify here will be used as the default installation source.



Note • *Files cached on the target system using this option will not be automatically removed when the user uninstalls your product.*

Path

Specify the directory in which files should be cached on the target system. You can enter a hard-coded path, as in **C:\Cached Files**, but it is recommended you include a destination variable from the drop-down list in the path, as in **[LocalAppDataFolder]Downloaded Installations**.



Note • *Only the destination variables included in the Path list are available for use as download locations.*

Release Configuration Panel



Project • *This panel is not available for InstallScript projects.*

The Release Configuration panel enables you to specify the type of compression, if any, that you would like to use in your release.

If you are rebuilding an existing release, the wizard remembers your previous build settings.

Table 12-8 • Settings on the Release Configuration Panel

Setting	Description
Compress all files	Select this option to compress all of your product's files into either a single executable file (**Setup.exe**) or a single Windows Installer package (.msi file).
Leave files uncompressed and separate from the installation package	Select this option if your product's files should not be compressed, and if they should be in a subfolder of the folder that contains your installation package.
Custom	Specify whether the release should be compressed: • **Compress all files**—Select this option to compress all of your product's files into either a single executable file (**Setup.exe**) or a single Windows Installer package (.msi file). • **Leave files uncompressed and separate from the installation package**—Select this option if your product's files should not be compressed, and if they should be separate from your installation package. • **Custom**—Select this option to compress only the files that are associated with one or more features into .cab files. If you select this option, the Release Wizard displays the Custom Compression Settings panel when you click Next to enable you to configure how the compression should be done.  ***Tip*** • *The output of the build process depends on your media type, compression, and spanning settings.*



Note • *The output of the build process depends on your media type, compression, and spanning settings.*

- *If you choose a Network Image media type and the* ***Compress all files*** *option, all your data files are compressed into* **Setup.exe** *or your .msi database, depending on whether you include the* **Setup.exe** *installation launcher.*
- *If you choose a CD-ROM, DVD-ROM, or Custom media type, along with the* ***Compress all files*** *option, each disk image contains a* **DataN.cab** *file that contains your data files.*
- *If you choose a CD-ROM, DVD-ROM, or Custom media type, along with the Custom option compression, the data files that you choose to compress are placed in cabinet files named* **ComponentName.cab**.
- *If you choose a Web media type, the build output is described on the Web Type panel of the Release Wizard.*

Custom Compression Settings Panel



Project • *This panel is not available for InstallScript projects.*

This panel prompts you to choose which features you would like to compress.

Panel Options

Features to Compress

All of the features included in this release are displayed here. Select the features that you would like to compress by selecting the check box next to each one.

A component or merge module can be associated with more than one feature. In such a case, the Release Wizard applies the compression settings of the *first* feature with which the component or module is associated. For example, if ComponentA belongs to both Feature1 (whose files are compressed) and Feature2 (whose files are uncompressed), the compression settings are applied as follows. If Feature1 is the first feature to be built into the release, ComponentA's files will be compressed into the Windows Installer package or **Setup.exe**.

Select All

Click this button to select all features for compression.

Clear All

Click this button to deselect all features, meaning that none of the features will be compressed into the installation.

Setup Launcher Panel



Project • *The Setup Launcher panel is available for the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The Setup Launcher panel lets you specify whether you want to create a **Setup.exe** setup launcher. It also lets you specify whether you want to include the redistributable for Windows Installer 3.1 or earlier with your installation. For information about Windows Installer redistributables, see Adding Windows Installer Redistributables to Projects.

Table 12-9 • Settings on the Setup Launcher Panel

Setting	Description
Create installation launcher (Setup.exe)	Use this option to specify whether to create a **Setup.exe** setup launcher for the current release. The **Setup.exe** setup launcher is required in some cases. For example, if you want to install the Windows Installer engine on a target system, your release must include a setup launcher. It is also required if you selected the option to display the Language dialog in the Setup Languages panel. To learn more about scenarios that require a setup launcher, see Creating a Setup Launcher. The entire setup package and all of your application's files are compressed into **Setup.exe** if you selected the **Compressed and included within the installation package** option on the Release Configuration panel for a Network Image or Web—One Executable media type.
Include MSI 3.1 Engine	If you want to include the Windows Installer engine installation so that it can be installed or upgraded on a target system if necessary, select this check box. As an alternative, you can add an InstallShield prerequisite for the Windows Installer to your project. For more information, see Adding Windows Installer Redistributables to Projects.
Suppress warning if the Windows Installer service cannot be upgraded	To suppress the warning dialog that is displayed if the Windows Installer service cannot be installed or upgraded on the target system, select this check box.
Delay engine reboot until after your setup installation completes	Select the **Delay engine reboot until after your setup installation completes** option to delay any reboot that may be required by the Windows Installer engine installation until your setup program has completed. Clear this option to allow the system to reboot immediately after installing or updating the Windows Installer engine, if necessary.

Windows Installer Location Panel



Project • *This panel is not available for InstallScript projects.*

This panel specifies where the Windows Installer engine installers (**InstMsiA.exe** and **InstMsiW.exe**) are located.

Panel Options

Download engine from the Web

Select this option to download the Windows Installer engine installers (if necessary) from the URL that is specified on the Setup Launcher panel.



***Tip** • This option is recommended if your installation will be downloaded from the Internet and you want to minimize the package size and download time. The engine will not be downloaded if the correct version is already present on the target machine.*

Extract engine from Setup.exe

Select this option to compress the selected Windows Installer engine installers into **Setup.exe**, to be extracted at run time, if necessary.



***Tip** • Use this option if the entire installation must be self-contained in **Setup.exe**. Note that the Download engine from the Web option results in smaller installations and shorter download time; however, this option provides for a completely self-contained installation.*

Copy from source media

Select this option to leave the Windows Installer engine installers on the root of the source media. This option is not available for Web media types or Network Image media types with all of your files compressed into **Setup.exe**.



***Tip** • Use this option if the installation will be run uncompressed from fixed media—CD, DVD, or local network.*

InstallShield Prerequisites Panel

The InstallShield Prerequisites panel is displayed if your project contains one or more InstallShield prerequisites.

Use this panel to specify where the InstallShield prerequisites are located. Available options are as follows.

***Tip** • If you want to use the locations that are specified for each individual InstallShield prerequisite, you must select the Following Individual Selections option for the InstallShield Prerequisites Locations setting on the Setup.exe tab in the Releases view. For more information, see Specifying a Run-Time Location for a Specific InstallShield Prerequisite.*

Table 12-10 • Available Options on the InstallShield Prerequisites Panel

Option	Description
Download prerequisites from the Web	Download all of the InstallShield prerequisite files included in your project (if necessary) from the URL specified in the InstallShield prerequisite (.prq) file for each prerequisite. ***Tip** • This option is recommended if your installation will be downloaded from the Internet and you want to minimize the package size and download time. An InstallShield prerequisite will not be downloaded if the correct version is already present on the target machine.*
Extract prerequisites from Setup.exe	Compress the InstallShield prerequisite files into **Setup.exe**, to be extracted at run time, if necessary. ***Tip** • Select this option if the entire installation must be self-contained in **Setup.exe**. Note that the **Download prerequisites from the Web** option results in smaller installations and shorter download time; however, the **Extract engine from Setup.exe** option provides for a completely self-contained installation.*
Copy prerequisites from source media	Store the InstallShield prerequisite files on the source media. ***Tip** • Use this option if the installation will be run uncompressed from fixed media—CD, DVD, or local network.*
Follow individual selections	Use the locations that are specified for each individual setup prerequisite's properties in the Redistributables or Prerequisites view. For more information, see Specifying a Run-Time Location for a Specific InstallShield Prerequisite.

***Note** • If an InstallShield prerequisite is added to a project as a dependency of another prerequisite, the location for the prerequisite dependency follows the location setting of the prerequisite that requires it.*

Digital Signature Panel

The Digital Signature panel enables you to specify digital signature information for your package and the files in your package. Digitally signing your application assures your end users that the code within your application has not been modified or corrupted since publication.

Table 12-11 • Settings on the Digital Signature Panel

Setting	Description
Certificate URL	Type a fully qualified URL—for example, **http://www.mydomain.com**. This URL is used in your digital signature to link to a site that you would like end users to visit to learn more about your product, organization, or company.
Digital certificate information	To specify the digital certificate that you want to use to sign your release, click the Browse button next to this setting. The Certificate Selection dialog box opens, enabling you to specify either the location of the .pfx file or information about the certificate store that contains the certificate. To learn more, see Certificate Selection Dialog Box.
Certificate password	If the .pfx file that you are using has a password, enter it. InstallShield encrypts the password and stores it in your project file (.ism). At build time, InstallShield uses the password to sign files with a .pfx file. If your certificate is protected by a password but you do not enter it in this setting, signing with a .pfx file fails. Note that if you configure your project to use a certificate that was imported with password protection into a store, Windows prompts for the password at build time when InstallShield is attempting to sign your project's files. The strong key protection that Windows uses does not permit InstallShield to provide the password to the cryptographic provider.

Digital Signature Options Panel

The Release Wizard includes the Digital Signature Options panel if you enter digital signature information on the Digital Signature panel.

Use the Digital Signature Options panel to specify which files in your installation should be digitally signed at build time.



Tip • *You can also configure digital signature information for a release on the Signing tab in the Releases view.*

Table 12-12 • Settings on the Digital Signature Options Dialog Box

Setting	Project Type	Description
Sign Windows Installer Package	Basic MSI, InstallScript MSI, Merge Module	If you want to sign your Windows Installer package (.msi file), select this check box.  ***Note*** • *You can sign the Windows Installer package only if version 2.0 or later is selected for the MSI Engine Version setting on the Setup.exe tab.*
Sign Media	InstallScript	If you want to digitally sign the media header file (**Data1.hdr**), select this check box.
Sign Setup.exe	Basic MSI, InstallScript, InstallScript MSI	If you want to sign your **Setup.exe** file, select this check box. This setting is applicable only to releases that meet the following criteria: • Single-file **Setup.exe** • Compressed files • Network image media type
Sign files in package	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want to sign any of the files in your release, select this check box and then use the **Include patterns and files** and **Exclude patterns and files** boxes to indicate which files should be signed.  ***Windows Logo*** • *All executable files (including .exe, .dll, .ocx, .sys, .cpl, .drv, and .scr files) in an installation must be digitally signed for the Windows logo program.*

Table 12-12 • Settings on the Digital Signature Options Dialog Box (cont.)

Setting	Project Type	Description
Include patterns and files	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the files and file patterns that you want to be digitally signed at build time. Note the following guidelines: • You can type directly in the box. As an alternative, you can click the Files button, which launches the Browse for file dialog box. This dialog box lists all of the static files that are currently in your project. It also lists file patterns such as *.dll, which you can select. • To indicate a wild-card character, use an asterisk (*). For example, if you want to sign all .exe files, specify the following: ***.exe** Using wild-card characters is especially helpful if you include dynamically linked files in your project and you want to sign all files that match a certain pattern. • Put each file and each file pattern on its own line, with each separated by a carriage return. • Note that the files and file patterns that should not be signed override any files and file patterns that should be signed. For example, if you specify *.exe in the **Include patterns and files** box and in the **Exclude patterns and files** box, InstallShield does not sign any .exe files.

Table 12-12 • Settings on the Digital Signature Options Dialog Box (cont.)

Setting	Project Type	Description
Exclude patterns and files	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify any files and file patterns that you do not want to be digitally signed at build time. Note the following guidelines: • You can type directly in the box. As an alternative, you can click the Files button, which launches the Browse for file dialog box. This dialog box lists all of the static files that are currently in your project. It also lists file patterns such as *.dll, which you can select. • To indicate a wild-card character, use an asterisk (*). For example, if you do not want to sign any .drv files, specify the following: ***.drv** Using wild-card characters is especially helpful if you include dynamically linked files in your project and you want to avoid signing any files that match a certain pattern. • Put each file and each file pattern on its own line, with each separated by a carriage return. • Note that the files and file patterns that should not be signed override any files and file patterns that should be signed. For example, if you specify *.exe in the **Include patterns and files** box and in the **Exclude patterns and files** box, InstallShield does not sign any .exe files.
Sign files that are already signed	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If any of the files in your project are already digitally signed, determine whether you want InstallShield to replace those existing digital signatures with the digital signature that you specify on the Digital Signature panel. Note that this affects only files that meet the requirements that are specified in the **Include patterns and files** and **Exclude patterns and files** boxes. • To use the digital signature information that you provided on the Digital Signature panel to sign a file instead of any existing digital signature information that is already included with the file, select this check box. • To leave the existing digital signature information intact for any files that are already signed, clear this check box. This check box is cleared by default.

Table 12-12 • Settings on the Digital Signature Options Dialog Box (cont.)

Setting	Project Type	Description
Sign files in their original location	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Determine whether you want InstallShield to sign your original files or just the files that are built into the release: • If you want InstallShield to sign a temporary copy of each file and then use that signed temporary copy to build a release, clear this check box. Note that if you clear this check box, InstallShield will not modify or sign your original files. • If you want InstallShield to sign your original files, select this check box. This check box is cleared by default.  ***Tip*** • *The benefit of selecting this check box for a Basic MSI or InstallScript MSI project is that it helps create one patch that updates both compressed and uncompressed versions of a release that contains originally unsigned files.*

Password & Copyright Panel

The Password & Copyright panel prompts you to activate password protection for your installation. It also lets you specify whether you want to use default information or custom information for the Version tab on the Properties dialog box, which is displayed when end users right-click **Setup.exe** and then click Properties.

Table 12-13 • Password & Copyright Panel Settings

Setting	Description
Password Protect Setup.exe	If you want to require that end users enter a password in order for **Setup.exe** to run, select this check box, and then enter the password in the Password box.
Password	Enter the password that end users must enter in order to run the installation.
Show Password Dialog Box During Setup Initialization	 ***Project*** • *This setting is available in InstallScript projects.* If you want to launch the password-checking code in the **OnCheckMediaPassword** event handler function's default code, select this check box. If you clear this check box, you must enter your own code in the script to request the password from end users and check it by calling **FeatureValidate**.

Table 12-13 • Password & Copyright Panel Settings

Setting	Description
Use Custom Version Properties	 ***Project*** *• This setting is available in the following project types:* • *Basic MSI* • *InstallScript MSI* To override the default information and display your company's copyright information, as well as other information, select this check box. The information is shown on the Properties dialog box, which is displayed when end users right-click **Setup.exe** and then click Properties.
Copyright Notice	 ***Project*** *• This setting is available in the following project types:* • *Basic MSI* • *InstallScript MSI* Enter the copyright information that you want to be displayed on the Properties dialog box, which is displayed when end users right-click **Setup.exe** and then click Properties.

.NET Framework Panel



Project *• This panel is available in the following project types.*

- *Basic MSI*
- *InstallScript MSI*

To include one or more versions of the .NET Framework in an InstallScript or InstallScript Object project, use the Objects view to add the Microsoft .NET Framework object to your installation.

The .NET Framework panel is where you add support for the 32-bit versions of the .NET Framework 1.0, 1.1, or 2.0.



Note *• To include .NET Framework 3.5, 3.0 SP1, 3.0, 2.0 SP1 (x86, x64, IA64), or 2.0 (only x64, IA64) redistributables in your project, use the Redistributables view to add the appropriate InstallShield prerequisite for the Microsoft .NET Framework to your project.*

For more information, see Adding .NET Framework Redistributables to Projects.

Table 12-14 • .NET Framework Panel Settings

Setting	Description
Include or set up .NET Framework	Select this check box to include the .NET Framework with your installation. If it is not available on the target system, you can indicate the location where to find it and specify installation details.
.NET Version	Select the version of .NET Framework that you want to install to the target system. ***Note*** • *To include .NET Framework 3.5, 3.0 SP1, 3.0, 2.0 SP1 (x86, x64, IA64), or 2.0 (only x64, IA64) redistributables in your project, use the Redistributables view to add the appropriate InstallShield prerequisite for the Microsoft .NET Framework to your project.* *For more information, see Adding .NET Framework Redistributables to Projects.*
Download from the Web	Select this option to download .NET Framework support from the URL that is specified in the Release property sheet's .NET and J# Framework URL property.
Extract from Setup.exe	Select this option to extract the .NET Framework from Setup.exe and install it to the target system.
Copy from source media	Select this option to leave the .NET Framework on the root of the source media. This option is not available for Web media types or Network Image media types with all of your files compressed into Setup.exe.
Show full user interface when installing .NET Framework	If you select this check box, the Microsoft .NET Framework (English) Setup wizard appears when dotnetfx.exe installs the .NET Framework on the target system. This wizard shows the progress of the .NET Framework installation. If you clear this check box, the InstallShield Wizard appears when dotnetfx.exe installs the .NET Framework on the target system. This InstallShield Wizard shows the progress of the .NET Framework installation.
Delay requested .NET reboots until after your installation	Select this check box to delay any prompts to reboot until after the installation is finished. The .NET Framework may not function until after this reboot. Therefore, it is strongly recommended that you not select this check box if the .NET Framework is used during the installation.

.NET Run-Time Options Panel



Project • *This panel is not available for InstallScript projects.*

Use this panel to specify additional command line parameters to **DotNetFx.exe** and select the .NET core language.

Panel Options

Command line to pass to DotNetFx.exe

In the edit field, type a command line that you want to pass to **DotNetFx.exe**.

.NET Core Language

If you selected .NET version 1.1 in the .NET Framework panel, you can specify one .NET core language you want to distribute. This is the language that is used while the .NET 1.1 core redistributable is installed. If you selected version 2.0, the language options are disabled since they are all included with this version of the redistributable. To launch the Redistributable Downloader Wizard, click **Download more Languages**.

.NET Language Pack Run-Time Options Panel



Project • *This panel is not available for InstallScript projects.*

Use this panel to select the .NET language packs you want to include with your setup. In addition, you can provide a command line to be used with **LangPack.exe**.

Panel Options

Command line to pass to LangPack.exe

Type a command line to pass to **LangPack.exe**.

.NET 1.1 Language Packs to Distribute

Select the check boxes next to the language packs that you want to distribute with your setup. The options available depend on the Microsoft language pack redistributables you have on your system. To download more, click **Download more languages** to launch the Redistributable Downloader Wizard.

Visual J# Run-Time Options Panel



Project • *This panel is not available for InstallScript projects.*

Use this panel to select Visual J# options that will be used if the Visual J# run-time components are not installed on the target system.

Note *• The version of Visual J# installed at runtime aligns with the version of the .NET Framework you selected in the .NET Framework panel.*

Panel Options

Include Visual J# run-time components

Select this option to include the Visual J# run-time components you select below

Download from the Web

Select this option to have the setup download the components from the URL that is specified in the Release property sheet's .NET and J# Framework URL property.

Extract from Setup.exe

Select this option to extract the J# run-time components from Setup.exe during the installation process.

Copy from source media

The components are copied from the installation's source media.

Command line to pass to redist

Type a command line to pass to the J# redistributable. Consult Microsoft for valid command-line parameters.

Make the installation of J# optional

Presents the option to the end user while the installation is running—via a dialog.

If it is optional, and the setup is silent, install J#

If you elect to make J# installation optional and the installation is run silently, the option cannot be presented to the end user. In this case, the J# run-time components are installed without requiring the end user to make a choice.

Advanced Settings Panel

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

The Advanced Settings panel enables you to set the release location and file name format. For installation projects, you can also specify whether you want to create Autorun and package definition files along with your release. You can optionally specify an earlier release build to optimize patch creation.

If you are rebuilding an existing release, the wizard remembers the build settings you last specified.

Panel Options

Release Settings

Table 12-15 • Release Settings Area on the Advanced Settings Panel

Setting	Description
Location	Enter the directory where you would like your release to be saved. You may navigate to this directory by clicking the Browse button. Alternatively, you may enter a path variable for your release location. To use a path variable in this field, you can either navigate to the directory your path variable points to, or you can enter the name of the path variable you would like to use, enclosed in angle brackets—for example, <MyVariable>.
Use long file names	If your release will be stored on media that supports long file names, select this check box. If long file names are not supported, or if you are uncertain what file name format your media supports, clear this check box.
Optimize size	If you want InstallShield to use the LZX method of compression when building this release's cabinet files, select this check box. To use MSZIP compression, clear this check box. The Cab Optimization Type setting on the Build tab for a release in the Releases view also lets you select LZX or MSZIP compression for the .cab files; in addition, this setting also lets you specify that you do not want to use any compression for the .cab files.  ***Important*** • *Using compression generally decreases the size of your compressed files, but the build process may take more time to complete. Depending on the number and size of the files being compressed, the LZX compression and the build may take hours to complete. Therefore, if you select this check box, it is recommended that your build machine have the latest hardware to minimize the time that it takes for the build to complete.* This setting is used only if your release is configured to compress some or all of its files into .cab files.
Use path variable test values	If you used test values for any of your path variables, you can set those variables to their actual values at this time.

Table 12-15 • Release Settings Area on the Advanced Settings Panel (cont.)

Setting	Description
Generate Autorun.inf	Choose this option if you are distributing your installation on a CD-ROM and want to support the AutoPlay feature, a Windows logo requirement. A text file called Autorun.inf containing the instructions to autoplay your installation will be created in the root of your disk images folder. You can edit this file to add additional AutoPlay options or to pass command-line parameters to **MsiExec.exe** or **Setup.exe**. AutoPlay will not work in the disk image folders on your development system, because it is enabled only in root drives. Note also that users can turn off AutoPlay on their own systems.
Generate Package Definition File	To create a package definition file (.pdf) that enables end users to run your installation as an SMS job, select this check box. If you select this check box, InstallShield creates a version 2.0 .pdf.
Build UTF-8 Database	If you want the Windows Installer database, along with any instance or language transforms, to be built using the UTF-8 encoding, select this check box. The UTF-8 encoding supports characters from all languages simultaneously, enabling you to mix and match, for example, Japanese and German, or Russian and Polish, both in text shown to end users and in file names and registry keys. These mixed languages work correctly regardless of the current language of the target system. However, some scenarios result in user interface issues. For example, if an end user specifies the `/qb` command-line option or uninstalls the product from Add or Remove Programs, Windows Installer uses very small fonts to display the user interface text in a UTF-8 database. If you clear this check box, InstallShield creates an ANSI database when you build your release. This option does not let you mix characters from languages in different code pages. This check box is cleared by default.

Patch optimization (Optional) Area

To make the smallest possible patches, file keys in the **File** table should be identical in the earlier and later .msi databases. The patch-creation process uses the **File** table keys to determine if two files are the same file. (The actual file names cannot reliably be used, since a package might contain more than one file with the same name, installed under different conditions.) If the previous package is specified here, InstallShield uses identical **File** table keys for identical files.

For more information, see Upgrade Considerations.

Previous Windows Installer package

Browse for the earlier release version of the .msi file.

Release Settings Summary Panel

If you want to change any settings, click the Back button until you come to the panel that you would like to change. You can then return to the Release Settings Summary panel and start your build.

Panel Options

Build the Release

Select this option to build this release when you click Finish. Progress information for the build is displayed in the Output window.



***Note** • The Build your Release option does not appear on the Summary panel if the Release Wizard was launched from Microsoft Visual Studio.*

General Options Panel



***Project** • The General Options panel is available in the following project types:*

- *InstallScript*
- *InstallScript Object*

The General Options panel enables you to do the following:

- Create a self-extracting executable file for distributing your installation.
- Pass command-line options to **Setup.exe**.
- Pass preprocessor variable definitions to the compiler.
- Select whether to place the compiled script file (.inx file) in a cabinet file.



***Project** • In an InstallScript object project, the Compiler Preprocessor Defines list and the Advanced button are available; none of the other options on this panel are available.*

Create a single file executable

Select this check box to create a self-extracting executable file. The File Name and Icon lists are enabled only if this check box is selected.

File Name

Type a file name for the self-extracting executable file or select a path variable whose value defines the password.

Icon

Optionally specify the fully qualified name of the file from which the executable file's icon is taken at build time; if no file is specified, a default icon is used. To specify a file, type an explicit path, select and append to a path variable, or click the browse button to open the Change Icon dialog box, in which you can click the Browse button to select a

file. By default, the icon with index 0 is used; to specify a different icon, either select an icon in the Change Icon dialog box or append the icon's index or resource ID (preceded by a minus sign) to the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100.

Compress compiled script (.inx) file into media

If the compiled script file (.inx file) should be placed in a cabinet file, select this check box. If the file should be placed uncompressed in the Disk1 disk image folder, clear this check box.

Setup Command Line

Optionally specify any command-line parameters you want to pass to **Setup.exe** when the installation is launched; type a command line or select a path variable.

Compiler Preprocessor Defines

Optionally specify any preprocessor variable definitions; type the definitions or select a user-created build variable. Preprocessor variable definitions that are specified here apply only to the current release; they are not used when compiling the script for other releases. Use the following format, with no spaces before or after equals signs or commas:

```
MYVARIABLE1=123,MYVARIABLE2
```

Such variables can be tested in the script by #if and #ifdef statements that control the flow of the script. When you are creating an InstallScript object, enter IFX_OBJECTS.

Entering the name of a preprocessor variable in this box defines the variable. For example, if you enter MYVARIABLE in this box, the script commands in the following #ifdef loop are executed:

```
#ifdef MYVARIABLE
   // Commands
#endif
```

After you add or change a preprocessor variable definition in this box, you must compile your installation project for the addition or change to take effect. To learn how, see Compiling Scripts.

Other Disk Files

This button is enabled only if there are files in the Other folder under Advanced Files in the Support Files/Billboards view and you selected a media format that produces multiple disk images—which is true for all formats other than Network Image.

When you click this button, the General Options - Other Disk Files dialog box opens. This dialog box lets you specify a location in the disk image folders for each file that is in the Other folder.

Advanced

When you click this button, the General Options - Advanced dialog box opens. This dialog box lets you specify a custom location for the built files, the space that is reserved in the disk images folders, and whether the installation performs MD5 checking.

Features Panel



Project • *This panel is not available for the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The Features panel allows you to specify which features are included in the built release.

Panel Options

Use the "Include in Build" property

If this option is selected, each feature whose Include in Build property is set to Yes is included in the built release and each feature whose "Include in Build" property is set to No is not included.

Specify the features

Lets you specify for each feature whether the feature is included in the built release. To toggle a feature's inclusion, click the check box next to the feature. A checked box indicates that a feature is included in the built release; an unchecked box indicates that a feature is not included.

You cannot toggle the inclusion status of InstallScript objects. Objects will always have the same inclusion status as their parent feature.

A disabled checked box indicates that the feature cannot be excluded because it is required by a feature that is included.

If you toggle a feature to included, all its parent, child, and required features will also become included, as well as all child features of its required features. If you toggle the feature to excluded, then its child features will become excluded as well. If this feature's parent has no more included features, then the parent will become excluded as well; a parent feature must have at least one child feature included for it to be included. If you exclude all but one of the visible child features of a required feature, the remaining visible child feature of the required feature is automatically included and disabled so that at least one child feature remains included.

Visible features (features whose Visible property is set to Yes) are displayed with a filled icon (); invisible features are displayed with an unfilled icon ().

Media Layout Panel



Project • *This panel is not available for Windows Installer–based or InstallScript MSI projects.*

This panel allows you to specify, for individual features or for all features, whether the features' files are stored in cabinet files or placed uncompressed in the disk image.

Panel Options

Cabinet File(s)

Store all features' files in cabinet files.



Note • *You can specify in a component's Compressed property whether the component's files are compressed or uncompressed when stored in a cabinet file.*

CD-ROM Folder(s)

Place all features' files uncompressed in the disk image. Files associated with a given feature are placed in the disk image in the folder specified in the feature's CD-ROM Folder property. If no folder is specified, that feature's files are placed in the root of the disk image.

Custom

If this option is selected, clicking the panel's Next button displays the Custom Media Layout panel, which lets you specify for individual features whether the feature's files are stored in cabinet files or placed uncompressed in the disk image.

Custom Media Layout Panel



Project • *This panel is not available for the following project types:*

- *Basic MSI*
- *InstallScript MSI*

This panel allows you to specify, for individual features or for all features, whether the features' files are stored in cabinet files or placed uncompressed in the disk image.

Panel Options

Features in cabinets

Lets you specify for individual components whether the component's files are stored in cabinet files or placed uncompressed in the disk image. To toggle a component's storage mode, click the check box next to the component. A checked box signifies that a component's files are stored in cabinet files. An unchecked box signifies that a component's files are placed in the disk image in the folder specified by the component's CD-ROM Folder property. If no folder is specified, that component's files are placed in the root of the disk image.

Select All

Selects all features in the **Features in cabinets** box.

Clear All

Deselects all features in the **Features in cabinets** box.

User Interface Panel



Project *• This panel is not available for the following project types:*

- *Basic MSI*
- *InstallScript MSI*

This panel allows you to specify the look and feel of your setup's end user dialog boxes.

Panel Options

Use project's default

If this option is selected, the end user dialog boxes are displayed with the skin that is selected in the Dialogs view's Skins folder, as shown in the Dialog Preview graphic.

Do not use any skin

If this option is selected, the end user dialog boxes are displayed in standard Windows style, as shown in the Dialog Preview graphic.

Use skin specified below

If this option is selected, the end user dialog boxes are displayed with the skin that you select in the list box, as shown in the Dialog Preview graphic. Note that the BlueTC skin uses GIF files and so under a 16-bit color setting does not look as good as the Blue skin, which uses bitmaps (and so is much larger).

Dialog Preview

Displays a sample of how the end user dialog boxes will appear as a result of the selections you made with the option buttons and list box.



Note *• The dialog boxes that are displayed by the following functions cannot be displayed with a skin; they appear the same regardless of whether you have specified a skin.*

- *AskYesNo*
- *EnterDisk*
- *MessageBox*
- *MessageBoxEx*
- *RebootDialog*
- *SdComponentDialog—the Available Disk Space dialog box*
- *SdComponentDialog2—the Select Subcomponents dialog box*
- *SdComponentDialogAdv—the Available Disk Space dialog box*
- *SdConfirmNewDir*
- *SdConfirmRegistration*
- *SdExceptions*
- *SdShowMsg*
- *SelectDir*

- *SelectDirEx*
- *SprintfBox*

Display small initialization dialog

If checked, the setup initialization dialog box is the small box that was shown by setups created with InstallShield Professional version 6.31 and earlier—unless the setup displays a security, Save And/or Run Setup, Choose Setup Language, or Qualifying Product(s) Detected dialog box, in which case the setup initialization dialog box is larger and is consistent with the rest of the end user dialog boxes, regardless of the state of this check box. If unchecked, the setup initialization dialog box is larger and is consistent with the rest of the end user dialog boxes.

If this check box is unchecked, the setup initialization dialog box (and the Choose Setup Language dialog box, if any) are not displayed until the startup graphic closes. To specify the length of time for which the startup graphic displays, set Setup.ini's [Startup] section's SplashTime value.

Internet Options Panel



Project • *This panel is available in InstallScript projects.*

The Internet Options panel enables you to specify various Internet-related options.



Note • *Any release can be run over the Internet, regardless of its media type.*

Table 12-16 • Settings on the Internet Options Panel

Setting	Description
Generate One-Click Install	To create a One-Click Install installation that end users can download from the Internet and install, select this check box. If you select this check box, InstallShield includes with your installation an external setup player (**Setup.ocx** file) that downloads and then launches the **Setup.exe** file with the appropriate command line. If you select this check box, the other settings on this panel are enabled. To learn more, see One-Click Install Installations in InstallScript Projects.
Create a default Web page for the setup	Select this check box to indicate that you want InstallShield to create a default Web page (.htm file) in the Disk1 folder.

Table 12-16 • Settings on the Internet Options Panel (cont.)

Setting	Description
Enter the URL to launch	Do one of the following to indicate what URL should be launched if you click the Run From Web command on the Build menu: • To launch the **Setup.htm** file that InstallShield creates at build time and places in the Disk1 folder, leave this box blank. • To launch a different Web page, type the URL in this box or click the browse button to select the Web file.  ***Tip*** • *If you enter a URL, do not include the name of the page, and do not include an ending forward slash. For example, if the full URL for the* **Setup.htm** *file is* ***http://www.mypages.com/setup/Setup.htm****, enter the following as the URL to launch:* **http://www.mypages.com/setup**. *When you click the Run from Web command on the Build menu, InstallShield launches the appropriate URL (http://www.mypages.com/setup/Setup.htm, in the aforementioned example).*

Update Panel



Project • *This panel is not available for the following project types:*

- *Basic MSI*
- *InstallScript MSI*

This panel allows you to specify the release format and the existing releases for which the current release can be run as an update.



Tip • *To create a setup that can be run on a system on which no version of your product is currently installed, select the Full option and the Non-version specific option.*

Panel Options

Full

Specifies that the current release is a full release.

Non-version specific

Enabled only if the Full option button is selected. Specifies that the current release can update any existing version of your application (as long as it was installed by a setup that was created with InstallShield Professional version 6.0 or later), or can install your application on a system on which no version of your product is currently installed.

Version specific

Enabled only if the Full option button is selected. Specifies that the current release can update only the versions of your application that are specified in the combo box.

combo box

Enabled only if the Version specific option is selected. Lets you specify the versions of your product to which the update can be applied. Type a semicolon-delimited list of version numbers (for example, 1.2.3;1.2.4) or select a list of version numbers from the list box. If you leave this field blank, the update can be applied to all earlier versions of your product.

Differential

Specifies that the current release is a differential release.

list box

Enabled only if the Differential option button is selected. Displays the existing releases to which the current project is compared when creating the new differential release. A file in the current project is excluded from the differential release if the same file (with the same date and time, size, and attributes) exists in each of the specified releases; otherwise the file is included in the differential release. (Files in a component whose Difference property is set to No are always included in the differential release.) You specify these releases by clicking the Import and Add buttons.

Also displays the version information for each release.



***Note** • If the setup is unable to determine a specified release's version information, that version of your product cannot be updated by the setup. If no version information is displayed for a release, select the release, click the Modify button, then select the **Specify the version information below** option and type or select a version number.*

Add

Enabled only if the Differential option is selected. Opens the Existing Media dialog box, in which you can specify a release that is in the current project.

Import

Enabled only if the Differential option is selected. Opens the Media File Properties dialog box, in which you can specify a release that is not in the current project.

Modify

Enabled only if the Differential option is selected and a single release is selected in the list box. Opens the Media File Properties dialog box, in which you can change the release selection and the version information that is associated with the release.

Remove

Enabled only if the Differential option is selected and one or more releases are selected in the list box. Removes the selected release from the list.

Objects

Enabled only if the Differential option is selected. Opens the Object Difference dialog box, in which you select the conditions for including InstallScript objects in your differential release.

Object Difference Dialog Box

This dialog box opens when you click the Objects button in the Release Wizard's Update panel. This dialog box allows you to you specify the conditions for including InstallScript objects in your differential media.

Dialog Options

Include All

Specifies that the differential build will include all objects that would have been included in a full (non-differential) build. (This excludes, for example, objects that are associated with a feature whose Include In Build property is set to No or that is deselected in the Release Wizard's Features panel.)

Include If Changed

Specifies that the differential build will include only those objects that would have been included in a full build and are absent from, or have an earlier version number in, all the specified comparison media.



Note • *If the current project and all specified comparison media include the InstallShield Object Installer object, the media builder compares the files packaged by the InstallShield Object Installer in the current project to the packaged files in the specified media. The media builder includes in the InstallShield Object Installer only those files that are absent from—or have an earlier date and time, different size, or different attributes in—all the specified comparison media.*

Exclude All

Specifies that the differential build will include no objects.

Postbuild Options Panel



Project • *The Postbuild Options panel is not available for the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The Postbuild Options panel enables you to copy disk image folders to a folder or FTP site, or execute a batch file, after the release build is complete.

Table 12-17 • Settings on the Postbuild Options Panel

Setting	Description
Upload release files to the FTP site specified below	To automatically distribute your release to an FTP site, select this check box and enter the FTP location. Also enter the user name and password, if applicable. • **FTP Location**—Specify the FTP URL for the location. If you need to distribute your release to a path outside the FTP default folder, use a double slash (//). For example, to distribute your release to a root-level folder called **myproduct**, where the URL of the FTP server is **ftp://ftp.mydomain.com**, enter **ftp://ftp.mydomain.com//myproduct** for the FTP location. • **User Name**—If a user name is required to upload to the FTP location, enter the user name. • **Password**—If a password is required to upload to the FTP location, enter the password.
Copy the built media files to the folder specified below	If you want to be able to automatically distribute your release to a folder, select this check box and specify the FTP URL for the location. Existing folders with the same names as copied folders are overwritten, but no folders are deleted. Specify the folder path by typing the path in the box, or click the browse button to browse to the location. If the media format of the selected release is a network image, which creates only one disk image folder, the contents of the disk image folder, rather than the folder itself, are copied. If you chose to create a self-extracting executable file, the executable file, rather than the disk image folders, is copied.
Execute the batch or executable file specified below after building the media files	To launch a batch file (.bat) or executable file (.exe) after the release has been built, select this check box and specify the path in the box. You can either type the path or click the browse button to browse to the file. In addition, you can select a path variable in the list and then type the rest of the path. If you select this check box, InstallShield sets environment variables with the same names (including the angle brackets) and values as the project's build variables. InstallShield also sets the environment variable <ISMEDIADIR>, whose value is the path to the folder in which the release's Disk Images, Log Files, and Report Files folders are created. You can refer to the value of an environment variable in a batch file by surrounding the variable name with percent signs (%); for example: set PATH = %<ISPROJECTDIR>%;%PATH% When the file is finished, InstallShield deletes the environment variables that it set.

Setup Best Practices Wizard

This wizard is invoked whenever you violate Setup Best Practices while adding files to a component. The wizard tells you which Best Practices you are not following and gives you the option to correct the action.

The wizard appears only if you have enabled active monitoring of Best Practices in the Options dialog.

Welcome Panel

The wizard is informing you that by adding files to a component you have violated one of the following Setup Best Practices:

Table 12-18 • Setup Best Practices Violations

Best Practice	Description
Components Should Not Contain Multiple EXEs, DLLs, OCXs, CHMs, or HLPs	Each component should contain only one "portable executable" file (an EXE, DLL, or OCX) or help file (CHM or HLP). Windows Installer components are designed such that all of the advanced settings and component properties, such as the code GUID, refer ideally to a single portable executable or help file. Place other EXEs, DLLs, OCXs, CHMs, or HLPs into new components.
A File Cannot Be Associated with More than One Component	This Best Practice is extremely important for shipping reusable components and facilitating resiliency. Microsoft recommends that no file be shipped in more than one component, "even across products, product versions, and companies."
Use Merge Modules	Merge modules contain all of the files, registry entries, and logic necessary to install a distinct piece of functionality. You should not distribute a file for which a merge module is available. Using merge modules also helps you comply with two related requirements—the Best Practice to avoid associating a file with more than one component and the Windows logo guideline not to ship any core components. If the Best Practices monitoring option is enabled, the Setup Best Practices Wizard alerts you whenever you try adding to a component any file that is part of the merge modules that InstallShield provides.

Panel Options

Please do not warn me of other Setup Best Practice conflicts

Selecting this option if you do not want the Setup Best Practices Wizard to notify you of Best Practices violations. You can reverse this action later in the Tools | Options panel.

Compliance Panel

This panel tells you which files violate the Setup Best Practices listed in the Welcome panel by placing a warning icon next to the file and citing the specific Best Practice rule violated.

Panel Options

Files

The list contains all of the files you added to the component and tells you which of them violates Best Practices. You can ignore the wizard's recommendations by clicking Finish (or Cancel to exit the wizard altogether), or you can correct the Best Practices conflict by clicking the Remove button.

Remove

Select a file and click the Remove button to prevent the file from being added to this component.

Static Scanning Wizard



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

In Basic MSI and InstallScript MSI projects, the value that is selected for a component's.NET Scan at Build setting affects how the Static Scanning Wizard scans the files in that component. To learn more, see Static Scanning.

The Static Scanning Wizard enables you to scan the files that are in your project for potential dependencies that they may require. This wizard scans all .exe, .dll, .ocx, .sys, .com, .drv, .scr, and .cpl files in your project and lets you add any detected dependencies to your installation.

The new files that added to your project are added to the same feature as the file that depends on them, thereby ensuring they get installed when needed.



Task ***To launch the Static Scanning Wizard:***

1. In the View List under **Additional Tools**, click **Dependency Scanners**.
2. Click the **Perform Static Scanning** button.

The following panels are associated with the Static Scanning Wizard:

- Welcome
- Filter Files
- Scanning Progress
- File Selection
- Scan Results
- Completing the Static Scanning Wizard

Welcome Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Static Scanning Wizard enables you to scan the files that are in your project for potential dependencies that they may require. This wizard scans all .exe, .dll, .ocx, .sys, .com, .drv, .scr, and .cpl files in your project and lets you add any detected dependencies to your installation.

Click the Next button to begin scanning your project's files for dependencies.

Filter Files Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Static Scanning Wizard may list as dependencies certain files that you do not want added to your installation. For example, common system files that are already present on target machines usually do not need to be reinstalled. To avoid having these files added to your installation when you run the scanner, select the **Filter files** check box on the Filter Files panel.

To learn how to customize the list of files that are excluded from scans, see Filtering Files in Dependency Scanners.

Scanning Progress Panel



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Scanning Progress panel is displayed when the Static Scanning Wizard is scanning all .exe, .dll, .ocx, .sys, .com, .drv, scr, and .cpl files in your project for dependencies. No files are added until you have confirmed the findings of the scan.

File Selection Panel

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The File Selection panel displays a list of possible files and merge modules that you may need to add to your project. Use this panel to select the ones that you want to include in your installation. For more information, see Reviewing Dependency Scanner Results.

Table 12-19 • File Selection Panel Settings

Setting	Description
File	Indicate which files you want added to your installation by selecting the appropriate check boxes. The meaning of each type of icon is as follows: • : This icon indicates that the file is a system or driver files. They are normally not redistributed as part of an installation, and they can potentially cause target systems to become inoperable. Ensure that these files are necessary before including them. • : This icon indicates a non-system file. To include the file in your project, select the corresponding check box. • : This icon indicates a merge module. To include the merge module in your project, select the corresponding check box. • : This is the icon for a .NET assembly in your project. • : This icon indicates a file that the Static Scanning Wizard has detected but is already in the project.
Deselect All	If you want to clear all of the check boxes, click this button. You can then manually select each check box that corresponds with a file or merge module that you would like to add to your project.
Select All	If you want to select all of the check boxes, click this button. You can then manually clear each check box that corresponds with a file or merge module that you do not want to add to your project.

Scan Results Panel

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Scan Results panel shows the dependencies that the wizard identified and that you selected to be added to your project.

To add these dependencies to your project, click the Next button. To exit the wizard without adding the dependencies, click the Cancel button. To review the list of potential dependencies again and add or remove any of them, click the Back button.

Completing the Static Scanning Wizard Panel



***Project** • This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

When the Static Scanning Wizard shows the Completing the Static Scanning Wizard panel, the wizard has added the selected dependencies to your project.

Click Finish to close the wizard and return to InstallShield.

System Search Wizard



***Project** • The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The System Search Wizard provides the Windows Installer capability to search for a particular file, folder, registry key, .xml file or .ini value on a target system prior to installation.



Task ***To launch the System Search Wizard:***

1. In the View List under **Behavior and Logic**, click **System Search**.
2. Do one of the following:
   - To create a new system search: Right-click in the grid and click **Add**.

- To edit an existing system search: Right-click the item that you want to edit and then click **Modify**.

The System Search Wizard consists of the following panels:

- Welcome
- What do you want to find?
- How do you want to look for it?
- What do you want to do with the value?

Welcome Panel



Project *• The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Welcome panel in the System Search Wizard. This wizard enables you to add or modify a system search in your project. Click Next to add or modify a system search.

What do you want to find? Panel



Project *• The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The **What do you want to find?** panel of the System Search Wizard is where you specify the type of item you are searching for and where to conduct that search on the target system. Available options are:

- File path, by searching folders
- Folder path, by searching all drives
- Folder path, by searching in a specific folder
- Folder path, by searching for a specific file

- File path, as specified by a registry entry
- Folder path, as specified by a registry entry
- Registry entry
- File path, as specified by an .ini file value
- Folder path, as specified by an .ini file value
- .ini file value
- File path, as specified by a component's key file
- Folder path, as specified by a component's key path
- XML file value

If you are searching for a registry entry that contains a file path or folder, the file or folder must exist on the target system in order for the search to be successful. Otherwise, the value from the search is not set to a property.

If you are searching for a file or folder, the full path to that item is stored in the property. If you are searching for a registry value, the data of that value is stored in the property. The same holds true for .ini files. For components, the key path to the component is in the property.

To get the property using script, see MsiGetProperty in the Windows Installer Help Library.

How do you want to look for it? Panel (Defining Your System Search Method)



Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The **How do you want to look for it?** panel contains settings that let you customize your search. The settings in this panel vary, depending on the type of search that you selected in the previous panel.

Table 12-20 • Customizing Your Search

Type of Search	Corresponding Settings That Need to Be Configured
File path, by searching folders	Specify file details and where to search for the file. To learn more, see Specify the file details Panel (File Search Options).

Table 12-20 • Customizing Your Search (cont.)

Type of Search	Corresponding Settings That Need to Be Configured
Folder path, by searching all drives	Specify the name of the folder and how many subfolder levels you want to be searched. To learn more, see How do you want to look for it? Panel (Folder Search Options).
Folder path, by searching in a specific folder	Specify the name of the folder and where on target systems to look. To learn more, see How do you want to look for it? Panel (Specific Folder Options).
Folder path, by searching for a specific file	Specify file details and where to search for the file. To learn more, see Specify the file details Panel (File Search Options).
File path, as specified by a registry entry	Specify details about the registry entry. To learn more, see How do you want to look for it? Panel (Registry Search Options).
Folder path, as specified by a registry entry	Specify details about the registry entry. To learn more, see How do you want to look for it? Panel (Registry Search Options).
Registry entry	Specify details about the registry entry. To learn more, see How do you want to look for it? Panel (Registry Search Options).
File path, as specified by an .ini file value	Specify details about the .ini file value. To learn more, see How do you want to look for it? Panel (.ini File Search Options).
Folder path, as specified by an .ini file value	Specify details about the .ini file value. To learn more, see How do you want to look for it? Panel (.ini File Search Options).
.ini file value	Specify details about the .ini file value. To learn more, see How do you want to look for it? Panel (.ini File Search Options).
File path, as specified by a component's key file	Specify the component identifier. To learn more, see How do you want to look for it? Panel (Component Search Option).

Table 12-20 • Customizing Your Search (cont.)

Type of Search	Corresponding Settings That Need to Be Configured
Folder path, as specified by a component's key path	Specify the component identifier. To learn more, see How do you want to look for it? Panel (Component Search Option).
XML file value	Specify file details and where to search for the XML file. To learn more, see Specify the file details Panel (File Search Options).

Specify the file details Panel (File Search Options)



Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you are configuring the **File path, by searching folders** type of system search, the **Folder path, by searching for a specific file** type of search, or the **XML file value** type of search, the following settings are available on the **Specify the file details and specifically where to search for the file** panel:

Table 12-21 • Specify the file details and specifically where to search for the file Panel Settings

Option	Description
File Name	Enter the full name and extension of the file or application that you want to locate.  ***Note*** • *When you enter a file name, the wizard enables the Details button. To modify your search to include any particular version, date, size, or language, click this button. To learn more, see File Details Dialog Box.*
Look In	Specify where on target systems you want Windows Installer to search.
Maximum number of subfolders to search	Specify the maximum number of subfolders that you want Windows Installer to search for the file on the target system.

How do you want to look for it? Panel (Folder Search Options)

Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you are configuring the **Folder path, by searching all drives** type of system search, the following settings are available on the **How do you want to look for it?** panel:

Table 12-22 • How do you want to look for it? Panel Settings

Option	Description
Folder Name	Enter the full name and extension of the folder that you want Windows Installer to find.
Look In	Specify where on target systems you want Windows Installer to search.
Number of subfolder levels to search	Specify the number of subfolder levels that you want Windows Installer to search on target systems.

How do you want to look for it? Panel (Specific Folder Options)

Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you are configuring the **Folder path, by searching in a specific folder** type of system search, the following settings are available on the **How do you want to look for it?** panel:

Table 12-23 • How do you want to look for it? Panel Settings

Option	Description
Folder Name	Enter the full name and extension of the folder you want to locate.

Table 12-23 • How do you want to look for it? Panel Settings (cont.)

Option	Description
Look In	Specify where on target systems you want Windows Installer to search. You can either specify a full path or select a path from a previous search. If you want to specify a full path, you can click the Browse button to select an existing directory or create a new one. If you want to specify a path from a previous search, select it from the drop-down list. Note that the list is empty if your project does not contain any other searches.
Number of subfolder levels to search	Specify the number of subfolder levels that you want Windows Installer to search on target systems.

How do you want to look for it? Panel (Registry Search Options)



Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you are configuring the **Folder path, as specified by a registry entry** type of system search, the **Folder path, as specified by a registry entry** type of system search, or the registry entry type of system search, the following settings are available on the **How do you want to look for it?** panel:

Table 12-24 • How do you want to look for it? Panel Settings

Setting	Description
Registry Root	Select the registry root where you want Windows Installer to search on target systems.
Registry Key	Enter the exact registry key that is associated with the item that you are locating. For example, to locate `HKEY_LOCAL_MACHINE\SOFTWARE\Adobe\Acrobat` on the target system, you would enter **SOFTWARE\Adobe\Acrobat** in this setting.  ***Tip*** • *Ensure that the syntax is correct by copying the correct key name from the Windows Registry Editor.*

Table 12-24 • How do you want to look for it? Panel Settings (cont.)

Setting	Description
Registry Value (Optional)	To search for a specific registry value, enter the registry value exactly as it appears in the Windows Registry Editor. ***Note*** • *If this setting is blank, the system search looks for the registry key's default value.*
Search the 64-Bit Portion of the Registry	A 64-bit target system typically has two HKEY_LOCAL_MACHINE\Software keys: • HKLM\Software, which is for 64-bit applications • HKLM\Software\Wow6432Node, which is for 32-bit applications If you want Windows Installer to check the 64-bit portion of the registry (HKLM\Software) on 64-bit target systems, select this check box.

How do you want to look for it? Panel (.ini File Search Options)

Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you are configuring the **File path, as specified by an .ini file value** type of system search, the **Folder path, as specified by an .ini file value** type of system search, or the **.ini file value** type of system search, the following settings are available on the **How do you want to look for it?** panel:

Table 12-25 • How do you want to look for it? Panel Settings

Option	Description
INI File Name	Specify the .ini file name as it should appear on target systems. The file must exist in the Windows folder.
INI Section Name	Specify the name of the section in the .ini file that contains the value that you want Windows Installer to find.
INI Key Name	Specify the name of the key that contains the value that you want Windows Installer to find..

Table 12-25 • How do you want to look for it? Panel Settings (cont.)

Option	Description
Read entire line	If you want Windows Installer to read the entire line in the .ini file, select this check box.

How do you want to look for it? Panel (Component Search Option)



Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you are configuring the **File path, as specified by a component's key file** type of system search or the **Folder path, as specified by a component's key path** type of system search, the System Search Wizard displays the **How do you want to look for it?** panel. This panel is where you specify the identifier of the component whose key path or key file is to be used for the search. The component ID must be enclosed in curly braces {}.

Specify the data that you wish to find in the XML file Panel (XML File Options)



Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

If you are configuring the **XML file value** type of system search, the System Search Wizard displays the **Specify the data that you wish to find in the XML file** panel. This panel is where you specify which .xml element you want to search for in the .xml file on the target system.

Table 12-26 • Specify the data that you wish to find in the XML file Panel Settings

Option	Description
XPath to XML Element (Example: Element/ SubElement)	Specify an XPath expression to identify the data in the XML file that you want to find. For examples of XPath expressions, see Using XPath Expressions to Find XML Data in an XML File.
Look For	Select the type of XML data that you want to find. You can search for the value of an attribute in the element that you specify, the contents of the element, or the existence of the element.

What do you want to do with the value? Panel



Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

In the **What do you want to do with the value?** panel, specify where you want to store the results of the search. If the search fails, the property is undefined. Available options are:

Table 12-27 • What do you want to do with the value? Panel Settings

Setting	Description
Store the value in this property	Select the public property that you want to use to store the location that the system search finds on target systems. To store the value in a non-predefined public property, enter the name of the property in this setting. The property must be a public property and therefore have an identifier that consists of uppercase letters.  ***Note*** • *You can add a new property to your project through the Property Manager view. Any new properties that you create must be public properties and have an identifier that consists of all uppercase letters.*

Table 12-27 • What do you want to do with the value? Panel Settings (cont.)

Setting	Description
Additional Options	Select the appropriate option: • **Just store the value in the property**—Store the value in the public property that you identify on this wizard panel. • **Use the property in an Install Condition**—Store the value in the public property that you identify in this wizard panel and use it in an install condition. If you select this option and then click the Finish button, InstallShield displays the Condition Builder, which enables you to specify the conditions that the installation must meet.

What do you want to do with the value? Panel

Project • *The System Search Wizard is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

In the **What do you want to do with the value?** panel, specify where you want to store the results of the search. If the search fails, the property is undefined. Available options are:

Table 12-28 • What do you want to do with the value? Panel Settings

Setting	Description
Store the value in this property	Select the public property that you want to use to store the location that the system search finds on target systems. To store the value in a non-predefined public property, enter the name of the property in this setting. The property must be a public property and therefore have an identifier that consists of uppercase letters. ***Note*** • *You can add a new property to your project through the Property Manager view. Any new properties that you create must be public properties and have an identifier that consists of all uppercase letters.*

Table 12-28 • What do you want to do with the value? Panel Settings (cont.)

Setting	Description
Additional Options	Select the appropriate option: • **Just store the value in the property**—Store the value in the public property that you identify on this wizard panel. • **Use the property in an Install Condition**—Store the value in the public property that you identify in this wizard panel and use it in an install condition. If you select this option and then click the Finish button, InstallShield displays the Condition Builder, which enables you to specify the conditions that the installation must meet. • **Use the property as the destination for a component**—Store the value in the public property that you identify in this wizard panel and use it as the destination for a component that you identify. This option is available only when you are searching for a folder path, a registry entry that contains a folder, an .ini file that contains a folder, a component whose key path is a folder, or an XML file.

Transform Wizard

The Transform Wizard walks you through the steps of creating and applying transforms for your installation projects. Transforms represent the difference between two similar installation projects. When you apply a transform to one project, you are updating or changing it to incorporate the changes between the two projects.

Task

To launch the Transform Wizard:

On the **Tools** menu, click **Create/Apply Transform**. It is not necessary to have a project open in InstallShield.

The following panels are associated with the Transform Wizard:

- Welcome
- Specify Files
- Validation Settings
- Suppress Error Conditions
- Specify Output File Name
- Summary
- Completing the Transform Wizard

Welcome Panel

The Transform Wizard walks you through the steps of creating and applying transforms to your setup project. A transform represents the differences between two setup projects. For example, network administrators may want to distribute different configurations of a product to the various departments in the company. As a result, you can create a transform for every configuration of the product, then apply the appropriate transform as needed.

Panel Options

Create a transform

Select this option to compare two similar setup projects and create a transform that represents the differences between them. This transform can then be applied at run time or before the final setup is built.

Apply a transform

Select this option to apply a transform to an existing Windows Installer setup. This may be useful for network administrators who will be customizing a product for everyone.

Specify Files Panel

If you are creating a transform, select the two MSI files that you would like to compare. If you are applying a transform, select the MSI file that you would like the transform applied to and the transform that will be applied.

Panel Options

Base package

This field will contains the path to the project that is currently open. If you want to change this value, enter the path to a different project or click the Browse button to navigate to one. If you are applying a transform, this field should contain the path to the file that you would like the transform applied to.

Target Package

This option is only available if you are creating a transform. Enter the path to the MSI file that you would like to compare to your base file, or click the Browse button to navigate to this file. The differences between these two files will be compared and a transform will be created that will update the base package to the target package.

Transform

This option is only available if you chose to apply a transform. Enter the path to the transform (*.mst) file that you would like to apply, or click the Browse button to navigate to that file.

Validation Settings Panel

The Validation Settings panel is displayed if you selected the Create a transform option on the Welcome panel of this wizard.

The Validation Settings panel prompts you for information on how you would like to validate your transform. These validation items must be met in order for the transform to execute.

Panel Options

Do not perform validation prior to applying the transform to the base package

Select this option if you do not want to perform any validation prior to applying the transform.

Validate the following prior to applying the transform to the base package

Select this option to check for certain conditions before the transform is applied. If these conditions are not met, the transform will not be applied.

Table 12-29 • Conditions for Applying a Transform to a Base Package

Option	Description
Default language must match base package	Select this option if the language of the transform should match the language of the package to which the transform is being applied.
Product must match base package	Select this option if you want your transform to be applied only to .msi files that have the same product code as the base package.
Platform must match base package	Select this option if the platform for which the base package is intended should match the target platform of the transform.
Upgrade Code must match base package	Select this option if you want the transform to be applied only if the upgrade code of the .msi file matches that of the transform.
Product Version	Select the version type that you would like to validate against. You can choose between a combination of major version, minor version, or upgrade version. You can also choose to not validate against the product version.

Table 12-29 • Conditions for Applying a Transform to a Base Package (cont.)

Option	Description
Version Relationship	Specify how you would like to validate the product version. The choices are outlined below: • **None**—Do not validate against the product version. • **Applied Version < Base Version**—Apply the transform only if the product version of the .msi file to which the transform is being applied is less than the version number of the base .msi file. • **Applied Version <= Base Version**—Apply the transform if the product version of the .msi file to which the transform is being applied is less than or equal to the version number of the base .msi file. • **Applied Version = Base Version**—Apply the transform only if the product version of the .msi file to which the transform is being applied is equal to the version number of the base .msi file. • **Applied Version >= Base Version**—Apply the transform only if the product version of the .msi file to which the transform is being applied is greater than or equal to the version number of the base .msi file. • **Applied Version > Base Version**—Apply the transform only if the product version of the .msi file to which the transform is being applied is greater than the version number of the base .msi file.

Suppress Error Conditions Panel

As your transform is being applied to the target MSI file, certain error codes may be generated. This panel gives you the option of suppressing those error codes so the end user will not see them.

Panel Options

Do not suppress error conditions when the transform is applied to the base package

Select this option if you do not want to hide the errors reported during the application of the transform.

Suppress the following error conditions when the transform is applied to the base package

Select this option to hide certain error codes from your end users. You can choose to suppress the following errors:

Table 12-30 • Suppressible Error Conditions

Error	Description
Adding an existing row	If your transform tries to add a row to the MSI database that already exists, an error is generated. Select this option to suppress that error.

Table 12-30 • Suppressible Error Conditions (cont.)

Error	Description
Deleting a row that does not exist	Select this option to suppress any errors that occur as a result of your transform trying to delete a row from the MSI database that does not exist.
Adding an existing table	Select this option to suppress errors caused by your transform trying to add a table to the MSI database that already exists.
Deleting a table that does not exist	Select this option to suppress any errors that occur as a result of your transform trying to delete a table from the MSI database that does not exist.
Updating a row that does not exist	If your transform tries to update a row to the MSI database that does not exist, an error will be generated. Select this option to suppress that error.
Transform and database code pages do not match	Select this option to suppress error codes caused by code pages in the transform not matching those in the target database.

Specify Output File Name Panel

This panel prompts you for the location and name of the transform (*.mst) file that will be created as a result of the comparison of your two MSI files.

Panel Options

Folder location and file name

If you are creating a transform, enter the path and file name of the transform file that you would like to create. If you are applying a transform, enter the path and file name of the new MSI package that you are creating as a result of applying the transform.

Summary Panel

The Summary panel displays all of the settings you selected throughout the Transform Wizard. If you need to change any of these settings, click the Back button until you reach the appropriate panel.

Completing the Transform Wizard Panel

The last panel of the Transform Wizard notifies you of the wizard's success. When you click the Finish button, you will be taken back to the IDE.

For more information on transforms, see Creating Transforms. This page provides an overview of transforms and links to pages that discuss how to create a transform and the different ways that transforms can be applied.



***Tip** • For more information about the errors that can occur when a transform is being applied to an .msi database, see the MsiDatabaseApplyTransform topic in the Windows Installer Help Library.*

Upgrade Validation Wizard

The Upgrade Validation Wizard performs a set of tests to determine if your installation will properly upgrade older versions of your installation.



***Project** • This wizard does not apply to the following project types:*

- *InstallScript*
- *InstallScript Object*



Task ***To launch the Upgrade Validation Wizard:***

On the **Build** menu, point to **Validate**, and then click **Upgrade Validation Wizard**.

The following panels are associated with this wizard:

- Welcome
- Settings
- Summary

Welcome Panel

The Upgrade Validation Wizard performs a set of tests to determine if your setup will properly upgrade older versions of your setup.

Click Next to continue.

Settings Panel

Specify the following basic configurations for the upgrade validation:

Table 12-31 • Settings Panel

Configuration	Description
Specify the latest version of your setup	Click the browse button to locate the file. ***Tip*** • *Think of the latest setup as the new image or version that you want to place on the target machine, and think of the previous setup as the existing image or version on the target machine before an upgrade.*
Specify any previous versions of your setup you want to validate against	Click Add to append to the existing list. You can add *.msi or *.exe Windows Installer Setup files. You can also specify multiple previous versions of your setup. To remove an item from the previous versions list, you must first select it in the list control and then click Remove.

When you click Next, validation begins.

Summary Panel

The Summary panel allows you to view the results from validation. You can also view the results externally by opening the corresponding log file stored in **<ISProjectDataFolder>\Upgrade Validation Log\Results.log**.

Note • *You can run validation again with different settings by clicking the Back button and changing your settings.*

The following table describes the icons next to each log file entry.

Table 12-32 • Icons

Button	Description
	This is the information notice icon.
	This is the error notice icon.
	This is the warning notice icon.

User Interface Wizard

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The User Interface Wizard enables you to add various predefined wizard pages to your Advanced UI or Suite/Advanced UI project. Examples include:

- A page that allows end users choose the installation directory for an .msi package
- A page that lets end users enter customer information and serial numbers
- A page that shows a feature tree, as well as feature descriptions and sizes, and allows end users to select which features to install

Task ***To launch the User Interface Wizard:***

1. In the View List under **User Interface**, click **Wizard Interface**.
2. In the **Wizard Interface** explorer, right-click **Wizard Pages** and then click **Add Predefined Page**.

The User Interface Wizard consists of the following panels:

- Predefined Task Pages Panel
- Task Configuration Panel

Predefined Task Pages Panel

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Predefined Task Pages panel is where you select the type of predefined wizard page that you want to add to your Advanced UI or Suite/Advanced UI project. This panel also lets you specify where in the sequence of wizard pages you want to schedule the new predefined page. You can resequence the wizard page later after adding it to your project.

The following table describes each of the types of available predefined wizard pages:

Table 12-33 • Types of Predefined Task Pages

Type of Wizard Page	Description
Browse for installation folder	This BrowseFolder wizard page lets end users select the installation directory for an .msi package. The Advanced UI or Suite/Advanced UI installation passes the path to the .msi package through the Windows Installer property **INSTALLDIR** when launching the package. This type of page is usually sequenced before an InstallationType or InstallationProgress wizard page.
Enter customer information for a Web Deploy package	 ***Edition*** • *Support for Web Deploy packages is available in the Premier edition of InstallShield.* This CustomerInformation wizard page lets end users enter information to publish a Web Deploy package, and pass the values to the package as **COMPUTERNAME**, **USERNAME**, **PASSWORD**, and **SITE**. This type of page is usually sequenced after the LicenseAgreement wizard page.
Enter customer information for an MSI package	This CustomerInformation wizard page lets end users enter customer information and serial numbers for an .msi package, and pass the values to the package through the Windows Installer properties **USERNAME**, **COMPANYNAME**, and **SERIALNUMBER**. This type of page is usually sequenced after the LicenseAgreement wizard page.
Enter installation's password	This Password wizard page is where end users enter the installation's password. If an end user does not enter the correct password, the installation exits. This type of page is usually sequenced at the beginning of an installation.  ***Note*** • *When you add this type of wizard page to your project, InstallShield adds a placeholder image control to it so that you can add a lock image to the page. Ensure that you either configure the control's Resource setting to indicate the file that you want to display with this control, or, to exclude an image, delete the image control.*

Table 12-33 • Types of Predefined Task Pages (cont.)

Type of Wizard Page	Description
Enter login information for a database server	 ***Edition*** *• SQL Support for suites is available in the Premier edition of InstallShield.* This SQLLogin wizard page lets end users enter database server login information (database server name, authentication credentials, database catalog name, etc.) in order to establish a connection to the database server that is referenced by one or more .msi packages in the suite. Values entered through these dialogs (or provided on a Suite command line) are passed to a selected .msi package command line with passwords marked as hidden.  ***Note*** *• When you add this type of wizard page to your project, you specify the database technology, driver, and the underlying Windows Installer properties and select the .msi package that receives these properties (because a suite may have multiple packages that need different SQL servers or else may need to share the same SQL server.)*
Maintenance welcome with update	The MaintenanceUpdateWelcome wizard page includes an update option to enable end users to download and install the latest available Advanced UI or Suite/Advanced UI update. To learn more, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.
View and accept supplemental license agreement	This LicenseAgreement wizard page requires end users to accept a supplemental end-user license agreement (EULA). The name of each EULA file must be unique; otherwise, the text and properties are linked. This type of page is usually sequenced after another LicenseAgreement wizard page.
View and accept supplemental license agreement (Scrolling required)	This LicenseAgreement wizard page requires end users to scroll through and accept a supplemental end-user license agreement (EULA). The name of each EULA file must be unique; otherwise, the text and properties are linked. This type of page is usually sequenced after another LicenseAgreement wizard page.

Table 12-33 • Types of Predefined Task Pages (cont.)

Type of Wizard Page	Description
View features and their size	This wizard page shows a feature tree, as well as feature descriptions and sizes that you have specified in your project. It allows end users to select which features to install. Note that this wizard page requires that you enter values in the Description and Cost settings of each feature in your Advanced UI or Suite/Advanced UI project. This type of page usually replaces the built-in InstallationFeatures wizard page.

Task Configuration Panel



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Task Configuration panel in the User Interface Wizard is where you specify additional information for the predefined wizard page that you are adding to your project. This panel is displayed for only some of the predefined wizard page types.

Table 12-34 • Required Information for the Task Configuration Panel

Type of Wizard Page	Description of the Required Information
Browse for installation folder	Select the package in the Advanced UI or Suite/Advanced UI project for which you want to allow end users to specify the path.
Enter customer information for an MSI package	Select the package in the Advanced UI or Suite/Advanced UI project for which you want to allow end users to specify customer information.
Enter installation's password	If you add this type of wizard page to your Advanced UI or Suite/Advanced UI project, the Task Configuration panel is not displayed.
View and accept supplemental license agreement	Specify the EULA file (.rtf) that you want to be displayed on the supplemental LicenseAgreement wizard page.

Table 12-34 • Required Information for the Task Configuration Panel (cont.)

Type of Wizard Page	Description of the Required Information
View and accept supplemental license agreement (Scrolling required)	Specify the EULA file (.rtf) that you want to be displayed on the supplemental LicenseAgreement wizard page.
View features and their size	If you add this type of wizard page to your Advanced UI or Suite/Advanced UI project, the Task Configuration panel is not displayed.

Visual Studio .NET Wizard for Visual Basic .NET, C# .NET, and Visual C++ .NET

The Visual Studio .NET Wizard for Visual Basic .NET, C# .NET, and Visual C++ .NET creates a new InstallShield installation project and adds it to a Microsoft Visual Studio solution.



Note • *The Visual Studio .NET Wizard is available only if Microsoft Visual Studio is installed on your system.*

When you are creating the InstallShield installation project, the Visual Studio .NET Wizard does the following:

- Creates a new InstallShield project (with the file name specified on the New Project dialog box) and adds it to the solution (.sln file).
- If you have the Scan at Build option set to Dependencies and Properties (on the .NET tab of the Options dialog box), the wizard adds all dependencies to your project at build time.
- Adds the primary outputs from every project in the solution to the InstallShield project.
- Updates the release settings to deploy the appropriate version of the .NET Framework via download.

To launch the wizard, select the appropriate project type in the New Project dialog box.

The following panels are associated with this wizard:

- Welcome Panel
- Solution Panel

Welcome Panel

The Visual Studio .NET Wizard enables you to add an InstallShield installation project to a Microsoft Visual Studio .NET solution.



Note • *The Visual Studio .NET Wizard is available only if Microsoft Visual Studio is installed on your system.*

Click Next to select a solution.

Solution Panel

In this panel, type the path or browse to the Visual Studio solution to which you want to add the InstallShield installation project.

Click Finish to create a new InstallShield project and add it to the selected solution.

Visual Studio Deployment Project Import Wizard

The Visual Studio Deployment Project Import Wizard enables you to import a Visual Studio setup or merge module project (.vdproj) into an InstallShield project (.ism). You can reuse this wizard if you want to import multiple Visual Studio projects into your InstallShield project.



Important • *If the Visual Studio setup or merge module project that you want to import into an InstallShield project contains one or more project outputs, the InstallShield project must be in the same Visual Studio solution that contains the Visual Studio setup or merge module project and all of its project dependencies.*



Task ***To launch the Visual Studio Deployment Project Import Wizard, do one of the following:***

- If you are using InstallShield by itself (without integration with Visual Studio): On the **Project** menu, click **Visual Studio Deployment Project Import Wizard**.
- If you are using InstallShield from within Visual Studio: On InstallShield toolbar, click the **Visual Studio Deployment Project Import Wizard** button.

This wizard consists of the following panels:

- Welcome
- Project File
- Options
- Summary

Welcome Panel

The Visual Studio Deployment Project Import Wizard enables you to import a Visual Studio setup or merge module project (.vdproj) into an InstallShield project (.ism). You can reuse this wizard if you want to import multiple Visual Studio projects into your InstallShield project.



Important • *If the Visual Studio setup or merge module project that you want to import into an InstallShield project contains one or more project outputs, the InstallShield project must be in the same Visual Studio solution that contains the Visual Studio setup or merge module project and all of its project dependencies.*

Click Next to begin using the wizard.

Project File Panel

The Project File panel is where you specify the Visual Studio project (.vdproj) that you want to import. The project can be a setup project or a merge module project.

Options Panel

The Visual Studio Deployment Project Import Wizard imports the project outputs, files, registry keys, file extensions, custom actions, target system searches, launch conditions, and prerequisites from your Visual Studio project into your InstallShield project. The Options panel is where you select which, if any, properties you also want InstallShield to import from the Visual Studio project into your InstallShield project.



***Note** • If you select the check box for an option, InstallShield overwrites the existing value in your InstallShield project with the value that is configured in the Visual Studio project. For example, if you select the Product Name check box, InstallShield overwrites the value of the Product Name setting of your InstallShield project with the value that is set in the Visual Studio project.*

Table 12-35 • Available Options for Importing

Option	Description
Product Name	If you want your InstallShield project to use the ProductName property that is configured in the Visual Studio project that you are importing, select this check box. The product name is configured in the General Information view of InstallShield.  ***Project** • This option does not apply to merge module projects.*
Product Version	If you want your InstallShield project to use the value of the ProductVersion property that is configured in the Visual Studio project that you are importing, select this check box. The product version is configured in the General Information view of InstallShield.

Table 12-35 • Available Options for Importing (cont.)

Option	Description
INSTALLDIR	If you want your InstallShield project to use the value of the DefaultLocation property that is configured for the application folder in the Visual Studio project that you are importing, select this check box. If you select this check box, InstallShield updates the value of the INSTALLDIR setting in the General Information view with the path that is configured in the Visual Studio project.  ***Note*** • *Visual Studio lets you specify a directory path that contains multiple formatted properties, such as [ProgramFilesFolder][Manufacturer]\[ProductName], for the application folder. Visual Studio projects use a directory custom action to resolve the path at run time. However, InstallShield does not support this type of directory path. Therefore, InstallShield resolves the path during the conversion process and uses the* ***INSTALLDIR*** *property for the path.*
Add or Remove Programs Properties	If you want your InstallShield project to use the Add or Remove Program properties (AddRemoveProgramsIcon, Manufacturer, Description, ManufacturerUrl, Author, SupportUrl, and SupportPhone) that are configured in the Visual Studio project that you are importing, select this check box. If you select this check box, InstallShield updates the values of the following settings in the General Information view with the values that are configured in the Visual Studio project: • Display Icon (In the Visual Studio project, this is the AddRemoveProgramsIcon property.) • Publisher (In the Visual Studio project, this is the Manufacturer property.) • Add or Remove Programs Comments (In the Visual Studio project, this is the Description property.) • Publisher/Product URL (In the Visual Studio project, this is the ManufacturerUrl property.) • Support Contact (In the Visual Studio project, this is the Author property.) • Support URL • Support Phone Number  ***Project*** • *This option does not apply to merge module projects.*

Table 12-35 • Available Options for Importing (cont.)

Option	Description
Summary Information Stream Properties	If you want your InstallShield project to use the Summary Information Stream properties (Title, Subject, Keywords, and TargetPlatform) that are configured in the Visual Studio project that you are importing, select this check box. If you select this check box, InstallShield updates the values of the following settings in the General Information view with the values that are configured in the Visual Studio project: • Title • Subject • Keywords • Processor type in the Template Summary setting (In the Visual Studio project, this is the TargetPlatform property.)  ***Project*** • *This option does not apply to merge module projects.*
Product Code	If you want your InstallShield project to use the value of the ProductCode property that is configured in the Visual Studio project that you are importing, select this check box. The product code is configured in the General Information view of InstallShield.  ***Project*** • *This option does not apply to merge module projects.*
Upgrade Code	If you want your InstallShield project to use the value of the UpgradeCode property that is configured in the Visual Studio project that you are importing, select this check box. The upgrade code is configured in the General Information view of InstallShield.  ***Project*** • *This option does not apply to merge module projects.*

Table 12-35 • Available Options for Importing (cont.)

Option	Description
All Users	If you want your InstallShield project to use the value of the InstallAllUsers property that is configured in the Visual Studio project that you are importing, select this check box. If you select this check box, InstallShield updates the value of the **ALLUSERS** property in the Property Manager view based on the value that is set in the InstallAllUsers property in the Visual Studio project.  ***Project*** • *This option does not apply to merge module projects.*

Table 12-35 • Available Options for Importing (cont.)

Option	Description
Project Language	If you want your InstallShield project to use the language that is selected in the Localization property of the Visual Studio project that you are importing, select this check box. If you are importing a Visual Studio project into an InstallShield project and the following conditions exist, InstallShield replaces the existing string entry values in your project with default string entry values for the language of your Visual Studio project: • You select this Project Language check box in the Visual Studio Deployment Project Wizard. • The language of your Visual Studio project does not match the default language in your InstallShield project. (In Visual Studio, the Localization property indicates the project's language. In InstallShield, the Default Language setting in the String Editor view indicates the project's default language.) • InstallShield has support for the language that is used in your Visual Studio project. (If you are using the Professional edition of InstallShield, you may not have support for the language that is used in your Visual Studio project.) For example, if you select this Project Language check box, if the language of your InstallShield project is Spanish, if the language of your Visual Studio project is German, and if you are using the Premier edition of InstallShield, InstallShield replaces the Spanish run-time strings in your project with the default German translations. Thus, if you edit a string entry value by revising a setting such as the Publisher setting in the General Information view, and then you indicate in the wizard that you want to import the language of a Visual Studio project, InstallShield overwrites the value of the Publisher setting—as well as values for other settings—with the default German string entry values. Therefore, if the project language is changed while you are importing your Visual Studio project, review the settings in the General Information view and the String Editor view after the import, and modify the string entry values if appropriate.  ***Project*** • *This option does not apply to merge module projects.*

Summary Panel

The Summary panel lets you review the settings that you specified in the Visual Studio Deployment Project Import Wizard. To change any of the settings, click the Back button until you reach the appropriate panel. To import the project, click the Finish button.

View Reference

The InstallShield installation development environment (IDE) is organized into views that incorporate various areas of functionality. The View Reference section describes each of those views in the InstallShield interface.

Installation Information View



Project • *The Installation Information view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

The Installation Information view contains links to other views that you can use to configure general settings about your installation.

General Information



Project • *The General Information view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

The General Information view contains basic information about your project, your company, and your product. Some of the information that you enter in this view is for your reference only, some (in Basic MSI, DIM, InstallScript MSI, and Merge Module projects) is necessary to comply with Windows logo requirements, and the rest is for setting basic project settings.

Update Notifications



Project • *The Update Notifications view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

In the Update Notifications view, you can add FlexNet Connect to your installation project. FlexNet Connect enables you to provide automatic application updates to your end users.

General Information View



Project • *The General Information view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

The General Information view contains basic information about your project, your company, and your product. Some of the information that you enter in this view is for your reference only, some is necessary to comply with Windows logo requirements, and the rest is for setting basic project settings.

When you create a new installation project, you must configure its General Information view settings. InstallShield creates new projects with default settings, but you should set your own values so that your project includes data that is specific to your needs.

The General Information view consists of the following elements:

- A row of buttons
- A grid of settings

The following table describes the buttons that are displayed in the General Information view.

Table 12-1 • Controls in the General Information View

Name of Control	Icon	Description
Categorized		Sorts the settings according to categories.
Alphabetical		Sorts the settings alphabetically.

For descriptions of each of the settings in the General Information view, see General Information View Settings.

General Information View Settings



Project • *The General Information view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

The General Information view settings are organized into the following main categories:

- General
- Summary Information Stream
- Add or Remove Programs
- Software Identification Tag

General Settings

Use the General area of the General Information view to specify details such as product name and product version. The following settings are available in this area.

Table 12-2 • General Settings

Setting	Project Type	Description
Project File Name	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	This read-only setting displays the path and name of the InstallShield project file. It also shows the type of project. The extension of the project file varies, depending on the project type: • **.ism project file**—InstallShield uses this extension for most project types, including Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module. • **.dim project file**—InstallShield uses this extension for DIM projects. • **.issuite project file**—InstallShield uses this extension for Advanced UI and Suite/Advanced UI projects. To learn more, see Project Types.
Project File Format	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Select the file format that you want to use for your project file (.ism or .dim). Available options are: • **Binary**—To save the .ism file as a database file, select this option. This format is best for the speed of opening and saving the project file. This is the default format for Basic MSI, DIM, InstallScript MSI, and Merge Module projects. If you select this project file format for these project types, you can modify the .ism file in a Windows Installer database editor. You can also modify the .ism file by using the Windows Installer API or its automation interface. • **XML**—To save the .ism file as a hierarchical text-based format, select this option. This project file format is best for use with source code control systems. It enables you to modify the project file using XML tools. This is the default format for InstallScript and InstallScript Object projects.  ***Note** • Both project file formats let you build releases from the command line.*

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Suite GUID	Advanced UI, Suite/Advanced UI	Enter a GUID that uniquely identifies the Advanced UI or Suite/Advanced UI installation. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. Since this code uniquely identifies your Advanced UI or Suite/Advanced UI installation, changing the Suite GUID after you have already distributed your release is not recommended.
Setup Languages	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/Advanced UI	 ***Project*** • *The behavior of this setting varies, depending on the project type.* In Basic MSI, InstallScript MSI, Merge Module, and Suite/Advanced UI projects, the Setup Languages setting lets you specify the languages that you want to be listed in the UI Languages setting in the Releases view. If a language is not listed for this Setup Languages setting at the project level, you cannot include that particular UI language in your project's releases. Advanced UI projects, which are available in the Professional edition of InstallShield, have support for only one language. Therefore, the Setup Languages setting in this project type is read-only. In InstallScript and InstallScript Object projects, the Setup Languages setting lets you specify the languages that you want to be listed in the Languages setting in the Components and Releases views in your project. In general, if a language is not listed for this setting at the project level, you cannot designate that a particular component in your project is targeted for that language; in addition, you cannot designate that the components and UI strings for a particular language are included in your project's releases. When you add a supported language to an Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, or Suite/Advanced UI project through this setting, InstallShield adds string entries for that language to your project. The string entries include the built-in user-interface string resources that are already translated. In DIM projects, the Setup Languages setting lets you specify the languages that you want the project to support. When you add a language to your project through this setting, InstallShield adds string entries for that language to your project. The string entries include string resources that you would need to have translated. For more information, see Selecting the Installation Languages.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Default Language	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	To override the default project language that is configured in the General Information view or the String Editor view, select the appropriate default user interface language for the selected release. For more information, see Setting the Default Project Language.
Platform Filtering	InstallScript, InstallScript Object	This setting lets you specify the platforms that you want to be available when you select operating system requirements for components or releases in your project. In general, if a platform is not listed for this setting at the project level, you cannot designate that a particular component or release in your project is targeted for that platform. To change the platforms for your project, click the ellipsis button (...) in this setting. The Platforms dialog box opens, enabling you to select the platforms for your project.  ***Note*** • *Specifying platforms at the project level does not create target system requirements for running the installation. If you want to create target system requirements in an InstallScript or InstallScript Object project, use the SYSINFO structure to identify the operating platform of the target system.*

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Maintenance Experience	InstallScript	Select the behavior that occurs when a target machine already has your product installed and the end user reruns your installation. Valid options are: • **Standard**—If the maintenance user interface should be displayed when a target machine already has your product installed and the end user reruns your installation, select this option. Only one entry for the product is listed in Add or Remove Programs. • **Multi-Instance**—If end users should be able to rerun an installation multiple times as a first-time installation rather than as a maintenance installation, select this option. Each time that an end user runs the installation, a separate entry is added to Add or Remove Programs. By default, this option lets end users install the product to a different location each time that they run the installation. This is useful for a product that an end user may want to install to different locations in order to experiment with different product configurations and then later remove only specific instances. The maintenance user interface is displayed only if an end user reruns the installation from Add or Remove Programs. • **No uninstall or maintenance**—If end users should not be able to uninstall the product or run it in maintenance mode, select this option. No entry is created for the product in Add or Remove Programs. To learn more, see Running an InstallScript Installation Multiple Times.
Enable Maintenance	InstallScript MSI	Indicate whether you want to display the full maintenance user interface (UI) or the uninstallation UI when end users rerun the installation on a system on which the product is already present. Available options are: • **Yes**—Unless the `/removeonly` command-line parameter is passed to **Setup.exe**, the system variable REMOVEONLY is set to FALSE when an end user reruns the installation, and the standard maintenance UI is displayed. • **No**—The system variable REMOVEONLY is set to TRUE when an end user reruns the installation, and the uninstallation UI is displayed. To learn more, see Configuring the Enable Maintenance Setting.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
InstallScript User Interface Type	InstallScript MSI	Specify the type of InstallScript user interface (UI) that you want to use for your installation. Valid options are: • **Traditional Style (Requires Setup.exe)**—If you want to use the InstallScript engine as an external UI handler for your InstallScript MSI installation, select this option. With this style, your installation must include a **Setup.exe** setup launcher. The setup launcher serves as a bootstrap application that initiates the InstallScript engine to display the UI and run the InstallScript code, and the Windows Installer to run the Execute sequence of the .msi package. This is the default option for this setting. • **New Style (Requires Windows Installer 4.5)**—If you want to use the InstallScript engine as an embedded UI handler for your InstallScript MSI installation, select this option. With this style, InstallShield embeds the InstallScript engine within the .msi package. The Windows Installer calls the InstallScript engine to display the UI. The Windows Installer also runs the Execute sequence of the .msi package. This option requires Windows Installer 4.5 on the target machine. This option also has some limitations that require careful planning if you decide to use this style. For detailed information about these two styles, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.
Product Name	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, MSI Database, MSM Database, Suite/ Advanced UI, Transform	Enter the name of the product, DIM, merge module, or object. For information on how the product name is used, see Specifying a Product Name. ***Project*** • *For InstallScript and InstallScript Object projects—Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value. (To use a path variable: On the Project menu, click Settings. Then select the appropriate path variable on the Application tab.)* *The product name is stored in the InstallScript system variable IFX_PRODUCT_NAME.* *For DIM projects—The value in this setting is not displayed or used at run time. You can use this setting in DIM projects as a reference for internally identifying different versions of the project.*

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Product Version	Advanced UI, Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, MSI Database, MSM Database, Suite/ Advanced UI, Transform	Enter the version number for your product. The version must contain only numbers. It is typically in the format *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535. Note that although you can include the fourth field (*ddddd*), the installation does not use this part of the product version to distinguish between different product versions. To learn more, see Specifying the Product Version.  ***Project*** *• For Basic MSI, InstallScript, and InstallScript MSI projects—If your release includes a* **Setup.exe** *file, the product version that you specify is displayed on the Properties dialog box for* **Setup.exe***. For more information, see Customizing File Properties for the Setup Launcher.* *For InstallScript and InstallScript Object projects—Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value. (To use a path variable: On the Project menu, click Settings. Then select the appropriate path variable on the Application tab.)* *For DIM projects—The value in this setting is not displayed or used at run time. You can use this setting in DIM projects as a reference for internally identifying different versions of the project. The value that you enter is displayed in the DIM References view of the project that contains this DIM.*
Product Code	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Transform	Enter a GUID that uniquely identifies this product. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. Since this code uniquely identifies your product, changing the product code after you have already distributed your release is not recommended. For more information, see Setting the Product Code in a Windows Installer–Based Project or Setting the Product Code in an InstallScript-Based Project.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Upgrade Code	Basic MSI, InstallScript MSI, MSI Database, Transform	Enter a GUID that can be used for your product's upgrade code. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. The upgrade code is a GUID that uniquely identifies the product family to which the product belongs. The upgrade code should be consistent across different versions and languages of a family of related products so that Windows Installer can use it to search for related versions of the product that are already installed. For more information, see Setting the Upgrade Code.
Install Condition	Basic MSI, InstallScript MSI, MSI Database, Transform	This setting lets you specify one or more conditions that must be true in order for your installation to run. For example, you can test for a specific operating system or minimum system requirements. If the conditions do not evaluate to True at run time, an error message is displayed and the product is not installed. To specify one or more conditions, click the ellipsis button (...) in this setting. For more information, see Setting Product Conditions. When you add a condition, InstallShield adds two new settings under the Install Condition setting: • **Condition**—This setting displays the conditional statement that you added. To edit the conditional statement, click the ellipsis button (...) in this setting. To delete the condition and its message, click the **Delete this condition** button in this setting. • **Message**—This setting displays the run-time error message that is configured to be displayed at run time if the corresponding condition is not met on the target system. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Exit Conditions	Advanced UI, Suite/Advanced UI	This setting lets you specify one or more exit error messages that you want the Advanced UI or Suite/Advanced UI installation to display under various conditions before ending the installation. For example, if your Advanced UI or Suite/Advanced UI installation requires Windows Vista or later, you could use the Exit Condition setting and its subsettings to specify an error message that you want to be displayed when end users launch your Advanced UI or Suite/Advanced UI installation on earlier versions of Windows. When end users dismiss the error message, the Advanced UI or Suite/Advanced UI installation ends. To specify an error message and one or more conditions, click the ellipsis button (...) in this setting.
Exit Conditions (continued)		When you add a condition, InstallShield adds two new settings under the Exit Conditions setting: • **Exit Message**—Use this setting to enter the error message that you want the Advanced UI or Suite/Advanced UI installation to display at run time if the conditions that are defined under this message are true. To delete the message and its conditions, click the **Delete this condition** button in this setting. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. • **Any/All/None**—This setting displays the conditional statement that you are defining. For more information, see Defining Exit Conditions for an Advanced UI or Suite/Advanced UI Installation.
Module Language	DIM, Merge Module, MSM Database	Select the language of the DIM or merge module, or select Language Independent.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
INSTALLDIR	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify the value of the Windows Installer property **INSTALLDIR**, which indicates the destination directory where most of the files of the product, DIM, or merge module will be installed at run time. Instead of hard-coding a path, you can enter a directory property as part of the path. To select a directory property, click the ellipsis button (...) in this setting. This enables you to select the appropriate directory from a list, or to create a new directory within a predefined directory. Separate further levels of subdirectories with a backslash—for example, **[ProgramFilesFolder]MyApp\Bin**.  ***Windows Logo*** • *To comply with the Windows logo program, your product's default destination should be a subfolder of the Program Files folder (***[ProgramFilesFolder]***), which can vary depending on the system's locale and user settings.*  ***Project*** • *In Basic MSI, InstallScript MSI, MSI Database, and Transform projects, the default value is as follows:* **[ProgramFilesFolder]***My Company Name**My Product Name* *The value that you enter in this setting is assigned to the property* ***INSTALLDIR****, which is the default destination folder for all of your product's features and components. For more information, see Setting the Default Product Destination Folder (INSTALLDIR).* *In DIM, Merge Module, and MSM Database projects, the default value is as follows:* **[TARGETDIR]** *In DIM, Merge Module, and MSM Database projects, if you would like to let the users of this DIM or module override the default destination directory, leave that default value in this setting. For more information, see Specifying the Default Destination Folder for a Merge Module.*
TARGETDIR	InstallScript, InstallScript Object	Specify the default value for the main target directory for your product. Typically this value is set to a value such as the following one: **<FOLDER_APPLICATIONS>\<IFX_COMPANY_NAME>\<IFX_PRODUCT_NAME>** For more information, see TARGETDIR.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Module ID GUID	Merge Module, MSM Database	This read-only setting displays a GUID that uniquely identifies the merge module. Whenever you create a new merge module, InstallShield generates a new GUID for it. The module ID GUID is appended to all merge module GUID foreign keys. For example, if you create a component in a merge module project, that component is listed in the Direct Editor and in the built .msm file as **ComponentName.MergeModuleGUID**. If you want to change this GUID, you can use the Direct Editor view to edit the ModuleID field of the **ModuleSignature** table. Note that if you change this value, you must also update the value in all of the tables in your project; you can do this from within the Direct Editor view.
Module Dependencies	Merge Module, MSM Database	This setting lets you specify one or more merge modules that are required for your merge module. To add one or more dependencies that are available in your redistributables gallery, click the ellipsis button (...) in this setting. The Module Dependencies dialog box opens, enabling you to select the modules that your merge module requires. To add a dependency that is not present on your machine, click the **Add a new module dependency** button, and then specify its version, language, and module GUID.
Name	Merge Module, MSM Database	This read-only setting shows one of the following values: • If you selected a dependency that is available in your redistributables gallery, this setting shows the name of that dependency. • If you chose to add a dependency that is not present on your machine, this setting indicates **[Merge Module Added by Setup Author]**. This setting is displayed only if you have added a merge module dependency to your project. If you want to remove this dependency from your project, you can click the **Delete this module dependency** button in this setting.
Version	Merge Module, MSM Database	If the merge module that you are creating requires a particular version of the dependency, specify the version number. If any version of the dependency is acceptable, you can leave the Version setting blank. This setting is displayed only if you have added a merge module dependency to your project.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Language	Merge Module, MSM Database	If the merge module that you are creating requires a dependency of a particular language, specify the language. If any language of the dependency is acceptable, select Language Independent. This setting is displayed only if you have added a merge module dependency to your project.
Module ID	Merge Module, MSM Database	If you selected a dependency that is available in your redistributables gallery, this setting indicates the module ID of that dependency. If you chose to add a dependency that is not present on your machine, enter the module ID of the dependency. The module ID must be in the following format: **ModuleName.ModuleGUID** For example, if the name of the merge module is **MyDependency** and the GUID is **{2560C1ED-E2F7-4FE6-A0E6-15A9DA4CE9B9}**, enter the following in this setting: **MyDependency.2560C1ED-E2F7-4FE6-A0E6-15A9DA4CE9B9** This setting is displayed only if you have added a merge module dependency to your project.
Module Exclusions	Merge Module, MSM Database	This setting lets you specify one or more merge modules that are incompatible with the merge module that you are creating. This may be necessary, for example, if an earlier version of your merge module is not compatible with your new module. To add one or more exclusions that are available in your redistributables gallery, click the ellipsis button (...) in this setting. The Module Exclusions dialog box opens, enabling you to select the modules that are incompatible with your merge module. To add an exclusion that is not present on your machine, click the **Add a new module exclusion** button, and then specify the requirements for the version, language, and module GUID.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Name	Merge Module, MSM Database	This read-only setting shows one of the following values: • If you selected an exclusion that is available in your redistributables gallery, this setting shows the name of that exclusion. • If you chose to add an exclusion that is not present on your machine, this setting indicates **[Merge Module Added by Setup Author]**. This setting is displayed only if you have added a merge module exclusion to your project. If you want to remove this exclusion from your project, you can click the **Delete this module exclusion** button in this setting.
Max. Version	Merge Module, MSM Database	Specify the maximum version number of the merge module that should be excluded. If you leave this setting blank, all versions after the value that you specify in the Min. Version setting are excluded. If you leave both the Max. Version and Min. Version settings blank, there is no exclusion based on version. This setting is displayed only if you have added a merge module exclusion to your project.
Min. Version	Merge Module, MSM Database	Specify the minimum version number of the merge module that should be excluded. If you leave this setting blank, all versions before the value that you specify in the Max. Version setting are excluded. If you leave both the Max. Version and Min. Version settings blank, there is no exclusion based on version. This setting is displayed only if you have added a merge module exclusion to your project.
Language	Merge Module, MSM Database	If you want to exclude merge modules that have a specific language or change the current value of this setting, click the ellipsis button (...) in this setting. The Exclusion Languages dialog box opens, enabling you to specify whether you want to exclude a particular language or all of the languages except a particular language. This dialog box also lets you select Language Independent, which indicates that the language should not be used to exclude a merge module. This setting is displayed only if you have added a merge module exclusion to your project.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Module ID	Merge Module, MSM Database	If you selected an exclusion that is available in your redistributables gallery, this setting indicates the module ID of that exclusion. If you chose to add an exclusion that is not present on your machine, enter the module ID of the exclusion. The module ID must be in the following format: **ModuleName.ModuleGUID** For example, if the name of the merge module is **MyExclusion** and the GUID is **{2560C1ED-E2F7-4FE6-A0E6-15A9DA4CE9B9}**, enter the following in this setting: **MyExclusion.2560C1ED-E2F7-4FE6-A0E6-15A9DA4CE9B9** This setting is displayed only if you have added a merge module exclusion to your project.
DIM GUID	DIM	This read-only setting displays a GUID that uniquely identifies the DIM. Whenever you create a new DIM, InstallShield generates a new GUID for it. The DIM GUID is appended to all DIM GUID foreign keys. For example, if you create a component in a DIM project, that component is listed in the Direct Editor as **ComponentName.DIM_GUID**. If you want to change this GUID, you can use the Direct Editor view to edit the ModuleID field of the **ModuleSignature** table. Note that if you change this value, you must also update the value in all of the tables in your project; you can do this from within the Direct Editor view.
Build Instructions	DIM	Enter any special instructions or comments about this project that may be helpful for the release engineers who are responsible for including this DIM in installation projects. These instructions are displayed in the DIM References view of the project that contains this DIM. The build instructions are not displayed to the end user.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Locked-Down Permissions	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Select the type of permissions that you want to use for securing files, folders, and registry keys for end users who run your product in a locked-down environment. Available options are: • **Custom InstallShield handling**—InstallShield uses a custom **ISLockPermissions** table and adds custom actions to your project to set permissions on the target system. This option is the default value. • **Traditional Windows Installer handling**—InstallShield uses the **LockPermissions** table in the .msi database to store permission information for your product. It is often more advantageous to use the custom InstallShield handling than the traditional Windows Installer handling. For example: • The custom option includes support for many well-known security identifiers (SIDs) that are not supported by the traditional option. • The custom option supports the use of localized user names for many well-known SIDs, unlike the traditional option. With the traditional option, if you try to use a localized name to set permissions on a non-English system, the installation may fail. • The custom option lets you specify that you want to deny a user or group from having the permissions that you are specifying. The traditional handling does not allow you to do this. That is, with the traditional handling, you can only set specific permissions; you cannot deny permissions. This is a project-wide setting that affects all new permissions that you set for files, folders, and registry keys in your project. If you have already configured some permissions in your project and then you change the value of this setting, InstallShield lets you specify whether you want to use the alternate handling method for those already-existing permissions. For more information about configuring this setting, as well as information about InstallScript support for setting permissions, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Show Per-User Option	Basic MSI	Specify whether you want to give end users the option of installing your product for all users or for only the current user. This run-time option is available only on Windows 7 and later systems, and on Windows Server 2008 R2 and later systems. Available options are: • **No**—Do not include the buttons that let end users specify how they want to install the product. • **Yes**—If the target system has Windows 7 or Windows Server 2008 R2, InstallShield adds buttons to the ReadyToInstall dialog. The buttons let end users specify how they want to install the product. If elevated privileges are required, the shield icon is included on the all-users button. If an end user selects the per-user button, the **ALLUSERS** property is set to 2, and the **MSIINSTALLPERUSER** property is set to 1. If an end user selects the all-users button, the **ALLUSERS** property is set to 1, and the **MSIINSTALLPERUSER** property is not set. If the target system has Windows Vista or earlier, or Windows Server 2008 or earlier, the ReadyToInstall dialog does not show the buttons that let end users specify how they want to install the product. The default value is No.  ***Note*** • *Selecting No does not prevent end users from setting* ***MSIINSTALLPERUSER*** *from the command line when they run your installation. If your installation does not support this, you may want to add a launch condition or other run-time check to prevent this from occurring.* To learn more, see Per-User vs. Per-Machine Installations.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Create MSI Logs	Basic MSI, InstallScript MSI, MSI Database, Transform	To specify whether Windows Installer 4.0 or later should log your installation, click the ellipsis button (...) in this setting, which launches the Logging Options for Windows Installer 4.0 and Later dialog box. This dialog box is where you specify whether Windows Installer should log your installation. You can also use this dialog box to customize the types of messages that are logged. There are three possible values for this setting: • **No**—Installations are not logged. This is the default value. • **Yes**—InstallShield populates the **MsiLogging** property with the default value of **voicewarmupx**. • **Custom**—InstallShield populates the **MsiLogging** property with the value that you specified on the Logging Options for Windows Installer 4 and Later dialog box. If the value of this setting is Yes or Custom, and if the installation is run with Windows Installer 4.0 or later, as well as Windows Vista or later or Windows Server 2008 or later, the following occurs: • The installer creates a log file according to the appropriate logging mode: either **voicewarmupx** (if the Create MSI Logs value is Yes) or whatever custom value you specified on the Logging Options for Windows Installer 4.0 and Later dialog box. • The installer populates the **MsiLogFileLocation** property with the log file's path. • A **Show the Windows Installer log** check box is added to the SetupCompleteSuccess, SetupCompleteError, and SetupInterrupted dialogs. If the end user selects that check box and then clicks Finish, the log file is opened in a text file viewer or editor.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Create MSI Logs (continued)		The Create MSI Logs setting applies to installations that are run with Windows Installer 4.0 or later on Windows Vista and later systems or Windows Server 2008 and later systems. The **Show the Windows Installer log** check box is not visible in run-time dialogs that are displayed on earlier systems that are running earlier versions of Windows Installer. ***Note*** • *If the value of this setting is Custom and you change it to Yes, any custom parameters that you defined for the* ***MsiLogging*** *property in the Property Manager will be overwritten with the default value. If you change it to No, InstallShield deletes any custom parameters from the* ***MsiLogging*** *property.* For more information, including details on how to customize the types of messages that are logged, see Specifying Whether Windows Installer Installations Should Be Logged.
Fast Install	Basic MSI, InstallScript MSI, MSI Database, Transform	If you want to reduce the time that is required to install a large Windows Installer package, consider selecting the check boxes for one or more of the following options: • No system restore point is saved for this installation • Perform only File Costing and skip checking other costs • Reduce the frequency of progress messages This setting configures the Windows Installer property **MSIFASTINSTALL**, which can be set at the command line. Windows Installer 5 includes support for this setting. Earlier versions of Windows Installer ignore it.
Company Name	InstallScript, InstallScript Object	Enter the name of your company. This value is used in the default script to set TARGETDIR (if the string entry COMPANY_NAME does not exist); it can be retrieved at run time by calling the **MediaGetData** function. ***Tip*** • *Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value. (To use a path variable: On the Project menu, click Settings. Then select the appropriate path variable on the Application tab.)*

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Executable File	InstallScript, InstallScript Object	Enter the name of the application's main executable file. This value can be retrieved at run time by calling the **MediaGetData** function. ***Tip*** • *Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value. (To use a path variable: On the Project menu, click Settings. Then select the appropriate path variable on the Application tab.)*
URL	InstallScript, InstallScript Object	Enter a product URL. This information is stored in the project file and is for your reference only. It is never displayed to the end user. ***Tip*** • *Instead of hard-coding a value, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value. (To use a path variable: On the Project menu, click Settings. Then select the appropriate path variable on the Application tab.)*
Server Locations	Transform	If you are storing the installation package and its related files for the product on a network server or Web site, click the ellipsis button (...) in this setting to specify the server or Web site locations. The locations should have the .msi package, as well as all of the files that might be required to, for example, repair the product or advertise a feature. The validity of the server location that you specify is determined when the installation needs to access the server remotely. That is, if a server is not available, or if you added an invalid server, the entry will be ignored if the resource is needed, and errors might be generated. Each location that is specified must have the complete source for the installation. The entire directory tree at each source location must be the same and must include all of the required source files, including any .cab files. Each location must have an .msi file with the same file name and product code. ***Tip*** • *Server paths can contain environment variables that are identified with a percent sign (%). For example, the following path uses the Home environment variable:* **\\Server1\%Home%\Office** The paths that you specify are stored in the **SOURCELIST** property. Windows Installer appends this list to the end of an end user's existing source list for the product at run time.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
Register Object	InstallScript Object	Specify whether you want the InstallScript object to be registered on your machine when you build a release for it. Registering an object on your development machine enables you to include the object in InstallScript projects that you create on that machine. ***Tip*** • *If the object is not registered when you build it, you can register it from within an InstallScript project. To learn more, see Registering Objects in InstallScript Projects.*
Object Wizard	InstallScript Object	Select the type of wizard, if any, that you would like to use for your object's default language. Available options are: • **No Wizard**—If your object does not require a wizard, select this option. • **Use InstallShield Object Stock Wizard**—If you want InstallShield to create a wizard based on the properties that you are creating for your object, select this option. With the stock wizard, all read-only properties are displayed but are not editable. All read/write properties can be changeable by the user. Write-only properties are not displayed. As an alternative, you can click the ellipsis button (...) in this setting if you want to select a custom wizard that you created for your object. To learn more about creating a custom wizard and using the InstallShield stock wizard, see Designing an Object's Wizard.
Register Custom Wizard	InstallScript Object	Specify whether you want InstallShield to register the custom object wizard for this object when you build a release for the object. ***Note*** • *This setting is applicable only if you specify a custom wizard in the Object Wizard setting.*
Font Registration	InstallScript, InstallScript Object	If you want all of the font files in your project to be registered on the target system at run time, select Enabled. Static file links have an individual setting for font registration as well. This setting must be used to set the behavior for dynamically linked font files.

Table 12-2 • General Settings (cont.)

Setting	Project Type	Description
DIFx Support (for 32-bit platforms)	InstallScript, InstallScript Object	To enable DIFx support in your project for installing device drivers on 32-bit systems, select Enabled. If you select Enabled, InstallShield adds the DIFxAPI libraries to your project when you build a release. For more information, see Installing Device Drivers.
DIFx Support (for 64-bit Itanium platforms)	InstallScript, InstallScript Object	To enable DIFx support in your project for installing device drivers on 64-bit Itanium systems, select Enabled. If you select Enabled, InstallShield adds the DIFxAPI libraries to your project when you build a release. For more information, see Installing Device Drivers.
DIFx Support (for 64-bit AMD platforms)	InstallScript, InstallScript Object	To enable DIFx support in your project for installing device drivers on 64-bit AMD systems, select Enabled. If you select Enabled, InstallShield adds the DIFxAPI libraries to your project when you build a release. For more information, see Installing Device Drivers.

Summary Information Stream

Windows Installer databases are implemented as COM structured storage, and COM structured storage files usually contain a Summary Information Stream. The Summary Information Stream contains information about your company and the product that is being installed.

The following settings are available in the Summary Information Stream area in the General Information view.

Table 12-3 • Summary Information Stream Settings

Setting	Project Type	Description
Title	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify the type of database that you are creating. For a product installation, the default value is **Installation Database**, which is the recommended value.  ***Project*** • *Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—The value that you enter is used on the Summary tab of the Properties dialog box that is displayed if you right-click the Windows Installer database and then click Properties.* *For DIM projects—The value in this setting is not displayed or used at run time. You can use this setting in DIM projects as a reference for internally identifying the project.* When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Subject	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter the name of the product.  ***Project*** • *In Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—The value that you enter is used on the Summary tab of the Properties dialog box that is displayed if you right-click the Windows Installer database and then click Properties.* *For DIM projects—The value in this setting is not displayed or used at run time. You can use this setting in DIM projects as a reference for internally identifying the project. The value that you enter is displayed in the DIM References view of the project that contains this DIM.* When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 12-3 • Summary Information Stream Settings (cont.)

Setting	Project Type	Description
Author	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify your company name.  ***Project*** • *In Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—The value that you enter is used on the Summary tab of the Properties dialog box that is displayed if you right-click the Windows Installer database and then click Properties.* *For DIM projects—The value in this setting is not displayed or used at run time. You can use this setting in DIM projects as a reference for internally identifying the project. The value that you enter is displayed in the DIM References view of the project that contains this DIM.* When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Keywords	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify keywords that describe the Windows Installer database for your product.  ***Project*** • *In Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—The value that you enter is used on the Summary tab of the Properties dialog box that is displayed if you right-click the Windows Installer database and then click Properties.* *For DIM projects—The value in this setting is not displayed or used at run time. You can use this setting in DIM projects as a reference for internally identifying the project.*

Table 12-3 • Summary Information Stream Settings (cont.)

Setting	Project Type	Description
Package Code	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database	This setting identifies the package code, which is the GUID for your installation package, DIM, or merge module. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting.  ***Project*** • *In Basic MSI, InstallScript MSI, Merge Module, MSI Database, and MSM Database projects—Because Windows Installer requires that any two .msi databases with identical package codes to have identical contents, you should change the package code before releasing a modified package. In the Releases view, you can use the Generate Package Code setting for a product configuration to specify whether you want InstallShield to generate a new package code every time that you build a release. The default value for this setting is Yes.* *For DIM projects—The value in this setting is not displayed or used at run time. You can use this setting in DIM projects as a reference for internally identifying the project. The value that you enter is displayed in the DIM References view of the project that contains this DIM.*
Template Summary	Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify the processor type and default language that your installation supports. List the processor type first, followed by your installation's default language, and separate them with a semicolon. If you have multiple entries in the language category, separate them with a comma. For example, if your installation runs only on Intel processors and English-based systems, enter **Intel;1033**. If your product runs on x64 processors and supports English and German, enter **x64;1033,1031**. For the language portion of this setting, use the number 0 if your installation is language neutral. Valid processor values include: • Intel • Intel64 • x64 Note that you can specify only one processor value. For more information, see Using the Template Summary Property. If the target machine does not meet the requirements that you specify for this setting, an error message is displayed and the installation exits.

Table 12-3 • Summary Information Stream Settings (cont.)

Setting	Project Type	Description
Summary Information Stream Comments	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter any comments about your product. A typical value for this setting is as follows: **This installer database contains the logic and data required to install MyProduct.**  ***Project*** • *In Basic MSI, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—The value that you enter is used on the Summary tab of the Properties dialog box that is displayed if you right-click the Windows Installer database and then click Properties.* *For DIM projects—The value in this setting is not displayed or used at run time. You can use this setting in DIM projects as a reference for internally identifying the project.* When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Schema	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting lets you specify the integer that identifies the minimum Windows Installer version that is required for your installation package or DIM. For a minimum of Windows Installer 2.0, enter **200**. For a minimum of Windows Installer 3.0, enter **300**. For a minimum of Windows Installer 3.1, enter **301**. For a minimum of Windows Installer 4.5, enter **405**. If the end user's system has a Windows Installer version earlier than the minimum requirement that you specify for the Schema setting—for example, if you specify a schema value of 405 because your installation uses Windows Installer 4.5 features, but an end user has Windows Installer 3.1—the installation displays an error message and exits. The value that you enter for the Schema setting is used for the Page Count Summary property of your Windows Installer database.  ***Note*** • *You can override this value for all of the releases that are associated with a particular product configuration in your project by setting the Schema setting for the product configuration in the Releases view.*

Table 12-3 • Summary Information Stream Settings (cont.)

Setting	Project Type	Description
Require Administrative Privileges	Basic MSI, MSI Database	Specify whether your .msi package requires administrative privileges for the execute sequence of your installation. The default is Yes. If you set this to No, InstallShield sets bit 3 in the Word Count Summary property to indicate that elevated privileges are not required to install the .msi package. Note that if you select No but your .msi package tries to perform a task for which it does not have adequate privileges, Windows Installer may display a run-time error. This setting applies to installations that are run with Windows Installer 4.0 or later on Windows Vista and later systems or Windows Server 2008 and later systems. Earlier versions of Windows Installer and Windows ignore this setting. Note that an end user's installation experience is more secure when installations are run with only the permissions that they need. Unless an application is designed to be run only by system administrators, it should be run with the least privilege. To learn how this setting and other InstallShield settings affect whether Windows Vista and later display a User Account Control prompt to elevate privileges, see Minimizing the Number of User Account Control Prompts During Installation.

Add or Remove Programs

Add or Remove Programs (which is called Programs in the latest versions of Windows) in the Control Panel provides end users with technical support links and telephone numbers, product update information, and information about a product's manufacturer. Depending on how the installation is configured, the end user may have the option of removing, repairing, or changing the installation with the click of a button. You can specify this information in your project by configuring the Add or Remove Programs settings in the General Information view.



Project • *In a Basic MSI project—To prevent your product from appearing in Add or Remove Programs, you can set the Windows Installer property* ***ARPSYSTEMCOMPONENT*** *to 1 in the Property Manager. Note that setting this property simply suppresses the display of your product in Add or Remove Programs. An end user can still remove your product by running the installation in maintenance mode or from the command line.*

In an InstallScript MSI project—To prevent your product from appearing in the Add or Remove Programs, select Yes for the Hide Add/Remove Panel Entry setting in the Releases view.

Table 12-4 • Add or Remove Programs Settings

Setting	Project Type	Description
Show Add or Remove Programs Entry	Advanced UI, Suite/Advanced UI	Indicate whether you want to show your Advanced UI or Suite/Advanced UI product's entry in Add or Remove Programs in the Control Panel. Available options are: • **Yes**—Your Advanced UI or Suite/Advanced UI product's entry is displayed on the target system in Add or Remove Programs. This is the default value. • **No**—Your Advanced UI or Suite/Advanced UI product's entry is not displayed on the target system in Add or Remove Programs. The end user cannot use Add or Remove Programs to remove the product, perform maintenance, or view support information. If you select this option, InstallShield disables the other Add or Remove Programs settings.

Table 12-4 • Add or Remove Programs Settings (cont.)

Setting	Project Type	Description
Display Icon	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	 ***Project*** • *The value that you should enter for this setting depends on what project type you are using.* **In Basic MSI, InstallScript MSI, MSI Database, and Transform projects:** Enter the path on your development system to the .ico, .exe or .dll file that contains the icon resource that you want to be used for your product's entry in Add or Remove Programs. Instead of manually typing the path and file name, you can click the ellipsis button (...) in this setting to browse to the file. **For InstallScript and InstallScript Object projects:** Enter the path on the target system to the .ico or .exe file that contains the icon resource that you want to be used for your product's entry in Add or Remove Programs. You can specify a path relative to a system variable that is enclosed by angle brackets—for example: **<TARGETDIR>\icon.ico** **In Advanced UI and Suite/Advanced UI projects:** Enter the path on your development system to the .ico, .exe or .dll file that contains the icon resource that you want to be used for your product's entry in Add or Remove Programs. Instead of manually typing the path and file name, you can click the ellipsis button (...) in this setting to browse to the file. If you want to use the icon in a file that is present on target systems (for example, in a file that is installed by one of the packages in the Advanced UI or Suite/ Advanced UI installation), select a directory property from the list in this setting, and then enter the rest of the path for the icon file. If you leave this setting blank, InstallShield uses the icon that is specified in the Setup.exe Icon File setting on the Setup.exe tab in the Releases view.
Display Icon Index	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Suite/ Advanced UI	If the icon file that you specify contains more than one icon resource, specify the index in this setting. • A nonnegative integer refers to the order of the icon resources in the executable file. For example, 0 refers to the first icon in the file, 1 refers to the second icon, and 2 refers to the third icon. • Use a negative number to refer to a specific resource ID. For example, the icon index -12 refers to the icon with a resource ID of 12.

Table 12-4 • Add or Remove Programs Settings (cont.)

Setting	Project Type	Description
Disable Change Button	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	Specify whether you want to disable the Change button for your product's entry in Add or Remove Programs. The Change button enables end users to change installation options after the product has been installed. End users can remove or add features as needed. ***Project*** *• In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.* *Also in an InstallScript project, you can specify a combined Change/ Remove button by setting the Disable Change Button and Disable Remove Button settings to Yes and setting the system variable ADDREMOVE_COMBINEDBUTTON to TRUE in your script before the* ***MaintenanceStart*** *function is called. (****MaintenanceStart*** *is called by the default code for the* ***OnMoveData*** *event handler.)*
Disable Remove Button	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	Specify whether you want to disable the Remove button for your product's entry in Add or Remove Programs. The Remove button enables end users to remove the product by clicking one button, which runs your uninstaller with a reduced user interface. ***Project*** *• For Basic MSI, InstallScript MSI, MSI Database, and Transform projects—If the end user clicks the Remove button to remove your product, actions in the User Interface sequence of the project are executed.* *In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.*
Disable Repair Button	Basic MSI, MSI Database, Transform	Specify whether you want to disable the Repair button for your product's entry in Add or Remove Programs. The Repair button enables end users to run the Windows Installer repair option if any files have been deleted or corrupted.

Table 12-4 • Add or Remove Programs Settings (cont.)

Setting	Project Type	Description
Publisher	Advanced UI, Basic MSI, InstallScript MSI, MSI Database, Suite/Advanced UI, Transform	Specify the name of the company that created the product. This information is displayed for your product's entry in Add or Remove Programs. The value that you enter is stored in the Windows Installer Manufacturer property. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Windows Logo*** • *If you want to meet the requirements of the Windows logo program, you must specify the publisher.*
Publisher/Product URL	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/Advanced UI, Transform	Enter a general URL for your company or product—for example, **http://www.installshield.com**. On some versions of Windows, the publisher name on the Support Info dialog box is a hyperlink to this URL. The Support Info dialog box is displayed when an end user clicks the support information hyperlink for your product's entry in Add or Remove Programs. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** • *In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.*

Table 12-4 • Add or Remove Programs Settings (cont.)

Setting	Project Type	Description
Support Contact	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Transform	Enter the name of the person or department that end users should contact for technical support. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Add or Remove Programs. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** • *In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.*
Support URL	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Transform	Enter the URL that you would like end users to visit for technical support information for your product. This URL is displayed for your product's entry in Add or Remove Programs. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Windows Logo*** • *If you want to meet the requirements of the Windows logo program, you must enter a valid URL.*  ***Project*** • *In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.*

Table 12-4 • Add or Remove Programs Settings (cont.)

Setting	Project Type	Description
Support Phone Number	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	Enter the technical support phone number for your product. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Add or Remove Programs. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** • *In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.*
Read Me	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	Enter the name or path of the Readme file for your product. As an alternative, you can link to a Readme file located on the Internet by specifying a valid URL. For more information, see Specifying a Readme File. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Add or Remove Programs. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** • *In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.*

Table 12-4 • Add or Remove Programs Settings (cont.)

Setting	Project Type	Description
Product Update URL	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	Specify a URL where end users can get information about product updates or download the latest version. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Add or Remove Programs. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** • *In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.*
Add or Remove Programs Comments	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	Enter comments about your product. This information is displayed for your product's entry in Add or Remove Programs. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** • *In an InstallScript project, the* ***OnMoveData*** *event handler writes the data for this setting to the target system's registry.*
English or Japanese IDE Settings	InstallScript Object	These areas—English IDE Settings and Japanese IDE Settings—let you specify information that should be available to InstallShield users when they view your object in the Objects view of the English and Japanese versions of InstallShield.
Use Default	InstallScript Object	If this setting is for a language other than the default language, specify whether you want to use the default language settings' values for this language. If you select Yes, the other settings under the Use Default setting become disabled. If you select No, the other settings under the Use Default setting become enabled so that you can specify language-specific values. This setting is disabled for the default language.
Display Name	InstallScript Object	Enter the name of your object as you would like it to appear in the Objects view.

Table 12-4 • Add or Remove Programs Settings (cont.)

Setting	Project Type	Description
Short Name	InstallScript Object	Enter a shortened version of your object's name, if desired. You can enter the same name that you entered for the Display Name setting.
HTML Help	InstallScript Object	Specify the object's help file, which must consist of a single Web file; this file is displayed in the right pane when the object is selected in the Objects view. Type an explicit path or path variable.
Icon File	InstallScript Object	The icon is displayed next to your object in the Objects view. The icon must be 16 pixels by 16 pixels with a maximum of 16 colors. The icon can be loaded from a DLL, .exe, or .ico file. When the file is a DLL or .exe, the icon index or resource ID can be specified after the path and file name separated by a comma. If a resource ID is specified rather than an index, it must be preceded by a dash (-). If no index is specified, the number 0 is assumed.

Software Identification Tag

Use the Software Identification Tag area of the General Information view to specify whether you want to include an ISO/IEC 19770-2 software identification tag in your installation. If a tag is included, this area also lets you specify the identification information that is not already specified in other areas of the General Information view. For more information, see Including a Software Identification Tag for Your Product.

If you include a software identification tag, you can include application model data for Microsoft System Center Configuration Manager in the tag. For more information, see Including Microsoft System Center Configuration Manager Application Model Data About Your Product.

Table 12-5 • Software Identification Tag Settings

Setting	Project Type	Description
Use Software Identification Tag	Basic MSI	Specify whether you want to include an ISO/IEC 19770-2 software identification tag in your installation. If you select Yes, use the other tag-related settings in this area of the General Information view to specify the identification information that is not already specified in other areas of the General Information view.
Require Software Entitlement	Basic MSI	Specify whether you want to require your product to have a corresponding software entitlement in order for software reconciliation to be considered successful. In general, if the software must be purchased, Yes should be selected for this setting; if the software is free, No should be selected for this setting.

Table 12-5 • Software Identification Tag Settings (cont.)

Setting	Project Type	Description
Unique ID	Basic MSI	Enter a unique ID that identifies the specific version of this specific product. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button (...) in this setting. Note that InstallShield uses the value that you enter as part of the name of the tag file (*TagCreatorID_UniqueID*.swidtag). Therefore, the ID that you enter must not contain any characters that are invalid for file names.
Tag Creator	Basic MSI	Enter the name of the organization that created the tag.
Tag Creator ID	Basic MSI	Enter the registration ID of the organization that created the tag. This ID helps to differentiate between different legal organizations that have the same creator name but are in different countries. The convention for the registration ID is as follows: **regid.YYYY-MM.ReversedDomainName,division** The registration ID consists of the following parts: • **regid.**—The string *regid* indicates that the XML portion is a registration ID for a software identification tag. A period (.) must be included after this string. • **YYYY-MM.**—This part of the registration ID identifies the first full month (MM) and the year (YYYY) in which the domain name was owned by the tag creator. For example, if you are creating the tag and you purchased the domain name February 15, 1999, you would use **1999-03** in this part of the registration ID, since the first full month the domain name was owned was March (03), and the year was 1999. The year and month must be separated by a dash. • **ReversedDomainName**—This part identifies the reversed domain name of the organization that is creating the software identification tag. For example, for the flexerasoftware.com domain name, the reversed domain name is: **com.flexerasoftware** • **,division**—This optional part starts with a comma (,), and is followed by an additional string. You can enter a string that helps to distinguish between different divisions or areas of the organization. If you do not want to use this optional distinguishing part of the registration ID, do not include the comma or an additional string in your entry. Note that InstallShield uses the value that you enter as part of the name of the tag file (*TagCreatorID_UniqueID*.swidtag). Therefore, the ID that you enter must not contain any characters that are invalid for file names.

Table 12-5 • Software Identification Tag Settings (cont.)

Setting	Project Type	Description
Software Creator	Basic MSI	Enter the name of the organization that created the software. This setting is optional. If you leave this setting blank, InstallShield uses the value of the Tag Creator setting for the name of the software creator.
Software Creator ID	Basic MSI	Enter the registration ID of the organization that created the software. This ID helps to differentiate between different legal organizations that have the same creator name but are in different countries. This setting is optional. If you leave this setting blank, InstallShield uses the value of the Tag Creator ID setting for the software creator ID. The convention for the registration ID is as follows: **regid.YYYY-MM.ReversedDomainName,division** The registration ID consists of the following parts: • **regid.**—The string *regid* indicates that the XML portion is a registration ID for a software identification tag. A period (.) must be included after this string. • **YYYY-MM.**—This part of the registration ID identifies the first full month (MM) and the year (YYYY) in which the domain name was owned by the tag creator. For example, if you are creating the tag and you purchased the domain name February 15, 1999, you would use **1999-03** in this part of the registration ID, since the first full month the domain name was owned was March (03), and the year was 1999. The year and month must be separated by a dash. • **ReversedDomainName**—This part identifies the reversed domain name of the organization that is creating the software identification tag. For example, for the flexerasoftware.com domain name, the reversed domain name is: **com.flexerasoftware** • **,division**—This optional part starts with a comma (,), and is followed by an additional string. You can enter a string that helps to distinguish between different divisions or areas of the organization. If you do not want to use this optional distinguishing part of the registration ID, do not include the comma or an additional string in your entry.

Table 12-5 • Software Identification Tag Settings (cont.)

Setting	Project Type	Description
Software Licensor	Basic MSI	Enter the name of the organization that owns the copyright for the software. This setting is optional. If you leave this setting blank, InstallShield uses the value of the Tag Creator setting for the name of the software licensor.
Software Licensor ID	Basic MSI	Enter the registration ID of the organization that owns the copyright for the software. This ID helps to differentiate between different legal organizations that have the same licensor name but are in different countries. This setting is optional. If you leave this setting blank, InstallShield uses the value of the Tag Creator ID setting for the software licensor ID. The convention for the registration ID is as follows: **regid.YYYY-MM.ReversedDomainName,division** The registration ID consists of the following parts: • **regid.**—The string *regid* indicates that the XML portion is a registration ID for a software identification tag. A period (.) must be included after this string. • **YYYY-MM.**—This part of the registration ID identifies the first full month (MM) and the year (YYYY) in which the domain name was owned by the tag creator. For example, if you are creating the tag and you purchased the domain name February 15, 1999, you would use **1999-03** in this part of the registration ID, since the first full month the domain name was owned was March (03), and the year was 1999. The year and month must be separated by a dash. • **ReversedDomainName**—This part identifies the reversed domain name of the organization that is creating the software identification tag. For example, for the flexerasoftware.com domain name, the reversed domain name is: **com.flexerasoftware** • **,division**—This optional part starts with a comma (,), and is followed by an additional string. You can enter a string that helps to distinguish between different divisions or areas of the organization. If you do not want to use this optional distinguishing part of the registration ID, do not include the comma or an additional string in your entry.

Table 12-5 • Software Identification Tag Settings (cont.)

Setting	Project Type	Description
Add SCCM App Model Data	Basic MSI	Specify whether you want to include application model data for Microsoft System Center Configuration Manager in the software identification tag in your installation. If you select Yes, configure this setting's subsettings as needed. For more information, see Including Microsoft System Center Configuration Manager Application Model Data About Your Product.
Application Type	Basic MSI	Select the type of application that your installation installs.
Supersedence	Basic MSI	If you want the installation that you are creating to supersede other installations, click the ellipsis button (...) in this setting. The Supersedence dialog box opens, enabling you to browse to the Windows Installer packages (.msi) that you want to be superseded.
Uninstall Superseded Applications	Basic MSI	Specify whether the applications that are being superseded should be uninstalled before installing the application via the current installation.

Update Notifications View

Project • *The Update Notifications view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*

For information on adding FlexNet Connect support to an InstallScript project, consult the Knowledge Base.

FlexNet Connect provides a mechanism that enables you to automatically notify your Web-connected end users when patches, updates, and product information for your application are ready for release.

Using FlexNet Connect is simple. When you enable FlexNet Connect, InstallShield includes the Software Manager in your installation. This desktop tool ships with your application and provides a tool for your end users to view available updates. To publish updates to your end users, you need to use a Web-based management portal called the FlexNet Connect Publisher site.

FlexNet Connect includes a variety of options that can be purchased together for a complete solution, or separately for a customized solution. To learn more, visit the Flexera Software Web site.

Organization View

Project • *The Organization view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

The first step in building anything is to lay a solid foundation. The base of your project is formed by specifying application information through the General Information view and—for installation projects—creating features in the Features view.

Setup Design



***Project** • The Setup Design view is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Transform*

The most important step in designing any setup is to lay out the various elements, the "building blocks" of the installation. You need to think both from your user's perspective as well as your own. The Setup Design view gives you a full look at your setup and all the features, components, and redistributables associated with it.

Features



***Project** • The Features view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Suite/Advanced UI*
- *Transform*

Features are the building blocks of your installation. They represent a distinct piece of your application—such as program files, help files, or clip art—to your end users. You can create features and subfeatures in the Features view.

Components



Project *• The Components view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Components are setup-authoring tools that help you organize similar application data, such as files, registry entries, and shortcuts, into logical groups. Unlike features, they constitute the developer's view of a project.

Setup Types



Project *• The Setup Types view is available in the following project types:*

- *InstallScript*
- *InstallScript MSI*

Setup types offer your customers multiple configurations of your applications. Generally, these configurations include Complete and Custom. Setup types allows your users to choose the best configuration for their needs. This view is available only for InstallScript and InstallScript MSI setup projects.

Packages



Project *• The Packages view is available in the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Use the Packages view to add packages and InstallShield prerequisites that you want to include in your Advanced UI or Suite/Advanced UI installation. This view is also where you define conditions for each package, associate each package with one or more Advanced UI or Suite/Advanced UI features, and configure other settings for each package.

DIM References



***Project** • The DIM References view is available in Basic MSI projects.*

Use the DIM References view to manage all of the DIM references in your project, view any specific instructions that are required for the referenced DIMs, and associate each referenced DIM with one or more features. This view also lets you schedule the DIM's custom actions and dialogs in the Basic MSI installation's sequences.

Setup Design View



***Project** • The Setup Design view is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Transform*

The most important step in designing any installation is to lay out the various elements, the building blocks of the installation. You need to think both from the end user's perspective, as well as your own.

You can design the entire installation hierarchy for your application visually under the Setup Design view. This is the only view in which you can associate components with features and subfeatures.

Features

Features are the building blocks of your application from the end user's perspective. They can be installed or uninstalled based on the end user's selections. Your entire application should be divided into discrete features that perform a specific purpose. Features can be created in both the Setup Design view and the Features view.

Components

Components enable you to group your application data together. Unlike features, components constitute the developer's view of a project—containing data such as files, registry entries, and shortcuts. Components are associated with features in the Setup Design view, and a component may belong to more than one feature. Components can be created in both the Setup Design view and the Components view.



***Tip** • The Components section contains the Files, Registry Entries, and Shortcuts nodes, in addition to the Advanced Settings for the component. If you do not want to see the subordinate nodes when they do not contain data, right-click Setup Design and select **Show Only Nodes with Data**. When this option is selected, only the nodes that contain data are displayed.*

DIMs

Developer installation manifest (DIMs) are separate pieces of an installation project that can be incorporated into a Basic MSI project to create a complete product installation. You can divide your product into separate, logical units to foster collaboration, enable distributed installation development, and enable reuse.

Redistributables

Redistributables enable you to install distinct, pre-existing pieces of functionality. For example, if your application requires Visual Basic run-time .dll files, you can include the Visual Basic Virtual Machine merge module, rather than add a file to your installation project and trying to determine its installation requirements.



***Note** • You can add redistributables—including InstallShield prerequisites, merge modules, and objects—to your installation in the Redistributables view (or the Prerequisites view in InstallScript projects and the Objects view in InstallScript and InstallScript Object projects).*

Files, Registry Data, and Shortcuts

Files, registry data, and shortcuts are elements that are added to a component to complete the hierarchy.

Advanced Settings

Advanced Component Settings let you handle installation of components with special requirements. By specifying the advanced settings, you can publish your component and register COM servers, file extension servers, MIME types, and so on. You can also use the component's advanced settings to create an application paths entry in the registry.

Features View



***Project** • The Features view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Suite/Advanced UI*
- *Transform*

A feature is the smallest installable part of a product, from the end user's perspective. It represents a specific capability of your product—such as its help files or a part of a product suite that can be installed or uninstalled based on the end user's selections. Your entire installation should be divided into features, each of which performs a specific purpose.

Subfeatures are further divisions of a feature. Because features should be self-contained elements of a product or product suite that an end user can selectively install, it might make sense for you to organize portions of your installation as subfeatures of a parent feature.



***Tip** • Although you can create many levels of subfeatures, you should keep the design as simple as possible for organizational purposes.*

Feature Settings



***Project** • Feature settings are available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *MSI Database*
- *Suite/Advanced UI*
- *Transform*

The settings in the Features view (and the feature settings in the Setup Design view) are organized into the following main categories:

- General
- Feature Events
- Run-Time Settings

General

When you select a feature in the Features view or the Setup Design view, the following settings are available in the General area:

Table 12-6 • General Settings for a Feature

Setting	Project Type	Description
Display Name	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	Enter the name that you want to be displayed for this feature on the CustomSetup dialog (or in Advanced UI or Suite/ Advanced UI projects, on the InstallationFeatures wizard page). When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** • *In Advanced UI and Suite/Advanced UI projects: When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.*

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
Description	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, MSI Database, Suite/ Advanced UI, Transform	Enter a brief description of the feature. A feature's description is displayed on the CustomSetup dialog (or, in Advanced UI and Suite/Advanced UI installations, on the InstallationFeatures wizard page) when the end user clicks a feature or subfeature. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** • *In Advanced UI and Suite/Advanced UI projects: When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.*
Remote Installation	Basic MSI, InstallScript MSI, MSI Database, Transform	Indicate the default install state for the feature. Valid options are: • **Favor Source**—The files in this feature are set to be run directly from the source medium, such as a CD-ROM or a network location, by default. • **Favor Local**—The files in this feature are set to be installed on the target system by default. • **Favor Parent**—The subfeature has the same default state as its parent feature. This option is available only if the feature is a subfeature. For more information, see Setting a Feature's Remote Installation Setting.  ***Caution*** • *If any of the feature's components contain a Windows service, select Favor Local. Although an end user could change the installation state through the CustomSetup dialog, the Windows Installer cannot install a service remotely.*

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
Destination	Basic MSI, InstallScript MSI, MSI Database, Transform	Select the folder on the target system into which the feature's files should be installed, or click the ellipsis button (...) to select or create a directory. ***Note*** • *If you want the destination to be configurable at run time, the destination folder that you select must be a public property (containing all uppercase letters).* For more information, see Setting a Feature's Destination.
Install Level	Basic MSI, InstallScript MSI, MSI Database, Transform	Enter an integer for this feature's install level. Unless the end user deselects features in the CustomSetup dialog, all features with an install level less than or equal to the value of the package's INSTALLLEVEL property are installed. You can change the INSTALLLEVEL property for the entire project in the Property Manager. The Install Level setting for a feature is compared against the **INSTALLLEVEL** property at run time to determine which features are available for installation. You can use this setting to create specific feature configurations. ***Project*** • *For InstallScript MSI installations, this setting is used only if no setup types are defined for the project.*
Display	Basic MSI, InstallScript MSI, MSI Database, Transform	Indicate how you want the feature presented to the end user in the CustomSetup dialog (in Basic MSI, MSI Database, and Transform projects) or the feature selection dialog (in InstallScript MSI projects). • **Visible and Collapsed**—The feature is displayed in the run-time dialog with its subfeatures collapsed by default. • **Visible and Expanded**—The feature is displayed in the run-time dialog with its subfeatures expanded by default. • **Not Visible**—The feature and subfeatures are not displayed in the run-time dialog. This setting does not have any direct impact on whether a feature is installed. A feature is not automatically installed if it is not visible—it just cannot be deselected if it would otherwise be installed, or selected if it should not be installed. For more information, see Displaying Features to End Users.

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
Visible	Advanced UI, InstallScript, Suite/ Advanced UI	Specify whether the feature should be visible on the feature selection dialog in InstallScript installations or the InstallationFeatures wizard page in Advanced UI and Suite/ Advanced UI installations. This setting does not have any direct impact on whether a feature is installed. A feature is not automatically installed if it is not visible—it just cannot be deselected if it would otherwise be installed, or selected if it should not be installed.  ***Note*** • *Since the only way for end users to change the selection status of an invisible feature is by selecting or deselecting a visible feature that requires the invisible feature, you should make every invisible feature a required feature for one or more visible features. Otherwise, end users will not be able to select or install features that are initially deselected.*
Advertised	Basic MSI, MSI Database, Transform	Select the appropriate advertisement option for this feature. Available options are: • **Allow Advertise**—End users have the ability to select the advertisement option for this feature in the CustomSetup dialog. Although advertisement is allowed, it is not the default option when the installation is run. • **Favor Advertise**—The feature is advertised by default. End users can change the advertisement option for a feature in the CustomSetup dialog. • **Disallow Advertise**—Advertising is not allowed for this feature. End users cannot elect to have the feature advertised in the CustomSetup dialog. • **Disable Advertise if Not Supported**—Advertisement works only on systems with Internet Explorer 4.01 or later. If the target system does not meet this criterion, advertising is not allowed. If the target system can support advertisement, advertising is allowed. For more information, see Advertising Features.

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
Required	Basic MSI, InstallScript MSI, MSI Database, Transform	Indicate whether the feature is required on the target system; if the feature is required, end users cannot deselect it in the CustomSetup dialog (in Basic MSI, MSI Database, and Transform projects) or the feature selection dialog (in InstallScript MSI projects). ***Project*** • *In InstallScript MSI projects, this setting is applicable to root-level features during a first-time installation. It is also applicable to subfeatures in an upgrade or patch. This setting is ignored for subfeatures during a first-time installation.*
Condition	Advanced UI, Suite/Advanced UI	This setting lets you specify one or more conditions that the Advanced UI or Suite/Advanced UI installation should use to evaluate whether the feature should be selected for installation by default on the InstallationFeatures wizard page. For example, if you want a particular feature to be selected by default on target systems that have a particular version of Windows, you can create a condition that specifies that version of Windows. Depending on how you configure the condition, you can have that Advanced UI or Suite/Advanced UI feature selected by default on the InstallationFeatures wizard page when an end user runs the Advanced UI or Suite/Advanced UI installation on that platform. Unless the end users deselects the feature, the Advanced UI or Suite/Advanced UI installation installs it. To add one or more new feature conditions, click the New Condition button in this setting. InstallShield adds a new row under the Condition setting. Select the appropriate option—All, Any, or None—from the list in this row. Then in this row, click the New Condition button, and select the appropriate option to continue building the conditional statement. If one or more conditional statements are configured, the Condition setting says **(Condition)**. If none are configured, the Condition setting says **(Empty)**. For descriptions of each of the condition settings, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
Release Flags	Advanced UI, Basic MSI, InstallScript MSI, Suite/ Advanced UI	If you want to use release flags to be able to selectively include and exclude this feature from different builds of your installation, enter one or more release flags to identify this feature. Separate multiple flags with a comma. For more information, see Assigning Release Flags to Features.
Condition	Basic MSI, InstallScript MSI, MSI Database, Transform	This setting lets you specify one or more conditional install levels. If a condition is true on the target system, the feature is given the corresponding install level. At run time, the feature's install level value is compared to the value of the global public property **INSTALLLEVEL** to determine whether the feature should be selected for installation by default. To specify one or more conditions, click the ellipsis button (...) in this setting. For more information, see Setting Feature Conditions. When you add a condition, InstallShield adds a new row to the grid of feature settings. The new row shows the install level for the feature, as well as the corresponding conditional statement that you added.
Comments	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Enter comments to help you track the changes you make to a feature, or for any future reference. The feature's comments are stored only in the project file and are not used in the installation at any time.
Required Features	InstallScript, InstallScript MSI, InstallScript Object	If one or more other features must be installed whenever the selected feature is installed, click the ellipsis button (...) and specify the other required features. For more information, see Using the Required Features Setting.
Allow UI Selection Change	Advanced UI, Suite/Advanced UI	Indicate whether end users should be allowed to select or deselect the feature on the InstallationFeatures wizard page. Available options are: • **Yes**—End users should be able to select or deselect the feature through the wizard interface. This is the default option. • **No**—End users should not be able to select or deselect the feature through the wizard interface. If this feature is visible on the InstallationFeatures wizard page, it appears in a read-only state (either selected or deselected, based on other factors, such as the feature's condition).

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
Follow Parent State	Advanced UI, Suite/Advanced UI	This setting is available for subfeatures; it is not available for root-level features. Indicate whether you want the subfeature's state to be determined by its parent feature. Available options are: ● **Yes**—The selected subfeature will be installed whenever its parent feature is installed. ● **No**—If the selected subfeature's parent feature is installed, the selected subfeature can be either installed or not installed. This is the default option. Note that if the selected subfeature's parent feature is not installed, the selected subfeature cannot be installed. This setting is especially useful if you want the subfeature to be hidden on the InstallationFeatures wizard page, but its parent feature is not hidden.
Status Text	InstallScript, InstallScript Object	Specify the status text for the feature. Your end users will see this text in the uppermost line of the progress indicator during the transfer of this feature. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
File Need	InstallScript, InstallScript Object	Indicate the severity of allowing end users to deselect the feature in a Custom setup type. Available options are: ● **Standard**—End users should be allowed to deselect the feature; no error messages are displayed. ● **Highly Recommended**—End users should be advised that the feature is highly recommended. ● **Critical**—End users should be warned that the feature should not be deselected. Your script can read this setting using the InstallScript function **FeatureGetData** with the FEATURE_FIELD_FILENEED constant. Depending on the return value, your script can display a message to alert the end user if a critical or highly recommended feature is deselected.

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
Include in Build	InstallScript, InstallScript Object	Specify whether the feature should be included in your release when you build it. Note that in the Features panel of the Release Wizard, you can override this setting for any of the features in your project.
Password	InstallScript	If you want to password-protect the feature, enter a password. You can specify a different password for each feature. You can also specify a password for the entire installation (in the Media Password setting on the Setup.exe tab in the Releases view). Note that the default script does not automatically validate a feature's password. You must modify the script to validate the password by inserting a dialog function (such as **AskText**) that prompts the end user to enter a password, and then calling the **FeatureValidate** function from within the script.  ***Caution*** • *Experienced hackers may be able to extract this password from your release. If security is a critical issue, it is recommended that you not depend solely on this password protection functionality.*

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
Encryption	InstallScript	Specify whether the files for the feature should be encrypted. Encrypted files can be read only by the InstallScript engine (or by an experienced hacker). If you plan to offer end users the option of running your product directly from the distribution media, do not encrypt the relevant files. InstallShield decrypts files only during installation to the end user's hard drive. Encryption is not the same as password protection. Encrypting a file scrambles it so that if someone opens it in a text editor, they cannot read strings and other data to get confidential information about your product. Password-protecting a feature prevents it from being installed by someone who does not know the password. You can encrypt a feature's files or you can password-protect the feature—or, for maximum security, you can do both. If a feature's files are encrypted but neither the feature nor the release is password-protected, the feature can be installed by anyone who obtains your distribution media (or a copy). If a feature is password-protected but its files are not encrypted, someone can open the files in a text editor and read confidential information—possibly including the feature's password.

Table 12-6 • General Settings for a Feature (cont.)

Setting	Project Type	Description
CD-ROM Folder	InstallScript, InstallScript Object	Specify the folder in the disk image in which the feature's files should be placed if the CD_ROM Folder(s) option is selected in the Media Layout panel of the Release Wizard, or if the feature's check box is selected on the Custom Media Layout panel of the Release Wizard. If either of these conditions are met but no folder is specified, the feature is placed in the root of the disk image or, if it is a subfeature, in the same folder as its parent feature. Folder names on CD-ROMs should conform to ISO 9660 (Level 1) standards. That is, folder names should be no more than eight characters long (with no extensions), and use only the characters A-Z, 0-9, and _ (underscore). By putting a folder structure on your CD-ROM, you can allow your product to be run from the CD-ROM. The folder structure on the CD-ROM does not need to mirror the folder structure that would be installed to a hard drive. All that is necessary is that your product be able to work with either folder structure. This functionality is typically used only for CD-ROM distribution; for other distribution media, you usually want to compress all features into .cab files.  ***Project*** • *For InstallScript Object projects, the value of this setting has no effect on where the object's files are placed when the object is included in an installation project.*
Included Components	InstallScript, InstallScript Object	This read-only setting lists the components that are associated with the feature. It is essential to an installation that components be associated with features; otherwise, no files will be installed. Use the Setup Design view to associate components with features. For more information, see Component-Feature Associations.
GUID	InstallScript, InstallScript Object	Enter a GUID that uniquely identifies this feature. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. The feature GUID identifies the feature in the installation's log file; if you change a feature's GUID in an upgrade, the InstallScript engine treats it as a new feature rather than an update to an existing feature.

Feature Events

When you select a feature in the Features view or the Setup Design view of an InstallScript, InstallScript MSI, or InstallScript Object project, the following settings are available in the Feature Events area:

Table 12-7 • Feature Event Settings for a Feature

Setting	Project Type	Description
OnInstalling	InstallScript, InstallScript MSI, InstallScript Object	This setting lets you specify an InstallScript function to be called before this feature is installed. Select the desired function from the list. Only exported functions appear in the list. Exported functions have this prototype: `export prototype FunctionName( );`
OnInstalled	InstallScript, InstallScript MSI, InstallScript Object	This setting lets you specify an InstallScript function to be called after this feature is installed. Select the desired function from the list. Only exported functions appear in the list. Exported functions have this prototype: `export prototype FunctionName( );`
OnUninstalling	InstallScript, InstallScript MSI, InstallScript Object	This setting lets you specify an InstallScript function to be called before this feature is uninstalled. Select the desired function from the list. Only exported functions appear in the list. Exported functions have this prototype: `export prototype FunctionName( );`
OnUninstalled	InstallScript, InstallScript MSI, InstallScript Object	This setting lets you specify an InstallScript function to be called after this feature is uninstalled. Select the desired function from the list. Only exported functions appear in the list. Exported functions have this prototype: `export prototype FunctionName( );`

Run-Time Settings

When you select a feature in the Features view or the Setup Design view of an InstallScript, InstallScript MSI, or InstallScript Object project, the following settings are available in the Run-Time Settings area:

Table 12-8 • Run-Time Settings for a Feature

Setting	Project Type	Description
FTP Location	InstallScript, InstallScript MSI, InstallScript Object	If you want to associate an FTP location with the feature, enter the FTP address in the following format: **ftp.domain.com** The address that you enter can be accessed from within the script using the **FeatureGetData** function with the FEATURE_FIELD_FTPLOCATION constant.
HTTP Location	InstallScript, InstallScript MSI, InstallScript Object	If you want to associate an HTTP address with the feature, enter the HTTP address in the following format: **http://www.domain.com** The address that you enter can be accessed from within the script using the **FeatureGetData** function with the FEATURE_FIELD_HTTPLOCATION constant.
Miscellaneous	InstallScript, InstallScript MSI, InstallScript Object	If you want to associate a text string with the feature, enter the text string. The text string that you enter can be accessed from within the script using the **FeatureGetData** function with the FEATURE_FIELD_MISC constant.

Components View



Project • *The Components view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Components are installation-authoring elements that help you organize similar application data, such as files, registry entries, and shortcuts, into logical groups. Unlike features, components constitute the developer's view of a project.

You can create and modify components in the Setup Design view (for installation projects) or the Components view.



Tip *• The Components explorer contains nodes for the files, registry data, and so on for each component. If you do not want to see the subordinate nodes when they do not contain data, right-click the top-level explorer and then click Show Only Nodes with Data. When this command is selected, only the nodes that contain data are displayed.*



Note *• To improve performance, the list displayed in the Files view truncates when there are more than a predefined number of files in a dynamically linked file with subfolders. **List Truncated** appears as the first item in the file list when this occurs. All files contained in the subfolders are built into your project.*

Component-Feature Relationships

Components are associated with features in the Setup Design view. For more information, see Associating New Components with Features.

Component Settings

For descriptions of each of the settings that are displayed when you click a component, see Component Settings.

Advanced Settings for a Component

For information about advanced settings that you can configure for a component, see Advanced Settings for a Component.

Component Settings



Project *• Component settings are available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The settings in the Components view (and the component settings in the Setup Design view) are organized into the following main categories:

- General

- Target Machine
- .NET Settings
- Run-Time Settings

General

When you select a component in the Components view or the Setup Design view, the following settings are available in the General area:

Table 12-9 • Component Settings in the General Area

Setting	Project Type	Description
Destination	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, MSI Database, MSM Database, Transform	Select the folder on the target system into which the component's files should be installed, or click the ellipsis button (...) to select or create a directory. To install the files to the product's default destination folder, select the appropriate location, depending on which project type you are using: • **[INSTALLDIR]**—in a Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, or Transform project • **<TARGETDIR>**—in an InstallScript or InstallScript Object project  ***Project*** *• In Basic MSI, InstallScript MSI, MSI Database, and Transform projects, if you want the destination to be configurable at run time, the destination folder that you select must be a public property (containing all uppercase letters).* *Note that in some project types (Basic MSI, InstallScript MSI, MSI Database, and Transform), you can also set the destination folder of a component's feature. For information on the effects of setting the destination for a feature as well as its components, see Component Destination vs. Feature Destination.* *In InstallScript and InstallScript Object projects, if you are installing files to WINSYSDIR64, you may need to select Yes for the component's 64-Bit Component setting. Otherwise, files being installed to WINSYSDIR64 may be redirected to SysWOW64, the 32-bit Windows system folder. For more information, see Targeting 64-Bit Operating Systems with InstallScript Installations.*

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Destination Permissions	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	To set permissions for the component's destination folder, click the ellipsis button (...) in this setting. ***Note*** • *Unless you need to designate specific permissions, it is recommended that you leave this setting undefined so that normal user permissions can apply to the destination folder.*
Component Code	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter a GUID that uniquely identifies this component. To have InstallShield generate a different GUID for you, click the Generate a new GUID button ({...}) in this setting.
Shared	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, MSI Database, MSM Database, Transform	***Project*** • *The behavior of this setting varies slightly, depending on whether the project type is Windows Installer based (Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, or Transform project) or InstallScript based (InstallScript or InstallScript Object project).* • **For a Windows Installer–based project**—Specify whether you want to mark this component's key file as shared. If the installation installs the key file and Yes is selected for its component's Shared setting, a reference count is created in the registry—if it does not already exist—and incremented. • **For an InstallScript-based project**—Specify whether you want to mark this component as shared. If the installation installs the component's files and Yes is selected for the component's Shared setting, a reference count for each of the component's files is created in the registry—if it does not already exist—and incremented. Note that the installation increments any existing reference count for any file in a component regardless of whether you mark the component as shared. However, if no reference count exists, the installation does not create one unless you select Yes for this Shared setting. For more information, see Managing Reference Counts for Shared Files.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Permanent	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify whether you want to mark this component as permanent. If you select Yes, none of the component's data (such as files, registry entries, and shortcuts) are removed from the target system when the component's feature is uninstalled. Validation rules require that you select Yes for any component being installed to **[SystemFolder]**.  ***Note*** • *If you are installing a font, it is recommended that you select Yes.*
Uninstall	InstallScript, InstallScript Object	Specify whether the component's files, registry entries, and other data should be uninstalled when the component's feature is uninstalled. This setting is useful for preventing the uninstallation of files that may be used by other products. If the component's files are not used by other products, the Uninstall setting should typically be set to Yes.
Condition	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting lets you enter a statement that the installation must evaluate before setting up your component's data on the target system. The component is not installed if its condition evaluates to false. However, the component is installed or advertised if its condition evaluates to true, assuming that its feature is selected for installation. To create a condition for the component, click the ellipsis button (...) in this setting. For more information, see Configuring Component Conditions.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Remote Installation	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Indicate the default install state for the component. Valid options are: • **Favor Source**—The files in this component are set to be run directly from the source medium, such as a CD-ROM or a network location, by default. • **Favor Local**—The files in this component are set to be installed on the target system by default. • **Optional**—The component uses the Remote Installation setting of its feature. For more information, see Setting a Component's Remote Installation Setting.  ***Caution*** • *If the component contains a Windows service, select the Favor Local option. Although an end user could change the feature's installation state through the CustomSetup dialog, the Windows Installer cannot install a service remotely.*

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
COM Extract at Build	Basic MSI, DIM, InstallScript MSI, Merge Module	Specify whether you want InstallShield to extract COM data from this component's key file at build time. Available options are: • **No**—InstallShield does not extract the COM information at build time. • **Yes**—InstallShield extracts the COM information every time that you build a release. If your COM server's interfaces are subject to change, this option is preferable to maintaining statically acquired or provided data in the COM Registration area under the component's Advanced Setting area. Select the No option if you want InstallShield to register the file according to the information that is statically contained in the COM Registration area under the component's Advanced Setting area. This advanced setting stores information about the file's COM classes, ProgIDs, and so on, that were either extracted in the Component Wizard or entered manually in the Advanced Setting area. ***Best Practice*** *• If you have marked a file for this component as self-registering, you should select No for the COM Extract at Build setting. Note, however, that calling self-registration functions is a violation of setup best practices.* To learn more about COM registration, see Extracting COM Registration Data at Build Time.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Languages	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	The Languages setting for a component enables you to specify the languages for which the component is intended, for use in build-time filtering. By default, components are language independent, meaning that none of the component's data (such as files and registry entries) are specific to a particular language. ***Project*** • *In InstallScript and InstallScript Object projects, if a language is not selected in the General Information view of a project, it is not listed as one of the available languages for the project's components.* To designate that the component is intended for only certain languages, click the ellipsis button (...) in this setting. The Languages dialog box opens, enabling you to select the appropriate languages for the component. For more information, see Marking Components as Language Dependent. ***Tip*** • *To learn how to specify which language-dependent components are installed at run time, see Installing Components Based on Language.*
Reevaluate Condition	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify whether you want the Windows Installer to reevaluate the component's condition when the product is reinstalled. This setting is useful for authoring transitive components—components that can replace each other when a product is reinstalled, for example, after an operating system upgrade. For more information, see Reevaluating Component Conditions During Reinstallation.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Never Overwrite	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify whether you want your installation to avoid overwriting a file if it already exists on the target system: • If you select Yes, the file—if it exists on the target system—is never overwritten, regardless of the file version. Selecting Yes for this setting overrides file versioning rules. • If you select No and the file version on the target system is newer than the version being installed, the file on the target system is not overwritten. However, if the version being installed is newer, the file on the target system is overwritten.  ***Important*** • *The Windows Installer checks for the existence of the component's key file when determining if it should install the component. If the component's key file does not exist on the target system, Windows Installer installs the component as if No were selected for the Never Overwrite setting. For more information on how the Windows Installer determines whether a component should be installed, see Overwriting Files and Components on the Target System.*
64-Bit Component	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, MSI Database, MSM Database, Transform	 ***Project*** • *The use of this setting varies, depending on whether the project type is Windows Installer based (Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, or Transform project) or InstallScript based (InstallScript or InstallScript Object project).*

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
64-Bit Component (cont.)		**For Windows Installer–Based Projects** Specify whether you want to mark this component as 64 bit. The component should be marked as 64 bit if any of the following are true: • The component contains files that need to be installed to a 64-bit location. • The component contains registry data that needs to be installed to a 64-bit area of the registry. • The component contains a 64-bit COM server and it needs to be self-registered. Note the following details: • If you include a 64-bit component in your installation, the installation cannot be run on 32-bit machines, and the Template Summary property must specify a 64-bit value. • If the component is 64 bit and it is replacing a 32-bit component, be sure to specify a new GUID for the component in the Component Code setting. For more information about 64-bit support, see Targeting 64-Bit Operating Systems with Basic MSI and InstallScript MSI Installations. **For InstallScript-Based Projects** Specify whether you want to mark this component as 64 bit. If you want your installation to install the files and registry data in this component to 64-bit locations on 64-bit systems and 32-bit locations on 32-bit systems, you can mark the component as 64 bit. For more information about 64-bit support, see Targeting 64-Bit Operating Systems with InstallScript Installations.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Source Location	Basic MSI, DIM, InstallScript MSI, Merge Module	Specify the name of a subfolder where the component's files should be stored in the source disk images if the component's files are not compressed. When you build a release, InstallShield copies the component's files to this subfolder in your disk image. This setting enables you to create unique folders for each component's files, which is necessary if you have different files with the same name in more than one component. If you do not specify a value for Source Location setting, you run the risk of overwriting files with the same name when you build an uncompressed release. This setting does not require a value, and in most cases, may be left blank. If you do enter a value, it must be a valid Windows folder name. For more information, see Installing Files of the Same Name.
Disable Registry Reflection	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Registry reflection keeps the 32-bit registry view and the 64-bit registry view in sync on the target machine. To enable registry reflection for all existing and new registry keys that are affected by the selected component, select No, which is the default value. To disable registry reflection on all existing and new registry keys that are affected by this component, select Yes. Windows Installer calls the **RegDisableReflectionKey** function on each key being accessed by the component. This function disables registry reflection for the specified key. Disabling reflection for a key does not affect reflection of any subkeys.  ***Note*** • *Only 64-bit systems with Windows Installer 4 or later support this setting. In addition, only Windows Vista and later and Windows Server 2008 and later support it. Other systems ignore this setting.* For more information about registry reflection, see Registry Reflection in the MSDN Library.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Multiple Package Shared Component	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	To enable shared component patching for the selected component, select Yes; otherwise, select No, which is the default value. If this multiple-package sharing feature were enabled in at least one package that is installed on the target system, Windows Installer 4.5 treats the component as shared among all of those packages. If a patch that shares this component is uninstalled, Windows Installer can continue to share the highest version of the component's files on the system. The purpose of this multiple-package component sharing is to prevent files from being downgraded during the uninstallation of a patch that contains a component that is shared with one or more other installed packages. The intent is to keep the highest version of the component's files present on the machine after uninstallation of that patch. The following scenario helps to illustrate the behavior: • Install product A with shared file 1.0.0.0. • Install product B with shared file 1.1.0.0. • Apply an uninstallable patch for product A with shared file 1.2.0.0. • Uninstall the patch for product A.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Multiple Package Shared Component (cont.)		Either of the following results may occur: • If the value of the Multiple Package Shared Component setting is Yes for either product A or product B and if Windows Installer 4.5 is present, uninstalling the patch from product A could restore version 1.1.0.0 to the target system. • If the value of this setting is No for product A and product B, the file would be downgraded to version 1.0.0.0, even though product B used 1.1.0.0. As a result, if product B requires version 1.1.0.0, product B may no longer work properly.  ***Note*** • *If the DisableSharedComponent policy is set to 1 on a target system, Windows Installer ignores this setting for all packages.* *Windows Installer 4.0 and earlier ignore this setting.* To learn more about this setting, see Specifying Whether Shared Component Patching Should Be Enabled for a Component.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Uninstall Superseded Component	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting determines how Windows Installer handles the selected component during installation of a superseding patch under certain conditions. It is designed to prevent the component's feature from being changed to an advertised state if a patch that is installed or uninstalled changes the component count for the feature. If the feature state changes to advertised, none of the feature's remaining components can be maintained. To specify that this component in the current patch should be flagged for uninstallation in order to avoid leaving this component orphaned on the target system after a superseding patch is applied, select Yes. If a subsequent patch is installed and it is flagged to supersede the first patch (that is, if Yes is selected in the Supersede column for the patch's family on the Sequence tab for the patch configuration in the Patch Design view), Windows Installer can unregister and uninstall this component if appropriate. If you select No, the superseding patch can leave an orphaned component on the target system. The default value for this setting is No. If you select No and the superseded patch installs a component for the product but the superseding patch removes that component, the component's feature state is changed to advertised, and it is not reinstalled. In addition, none of the remaining components associated with that feature can be maintained. If you select Yes, Windows Installer unregisters and uninstalls the component when the superseding patch is applied.  ***Note*** • *Windows Installer 4.5 includes support for this Uninstall Superseded Component setting. Earlier versions of Windows Installer ignore it.*  ***Tip*** • *Setting the* ***MSIUNINSTALLSUPERSEDEDCOMPONENTS*** *property has the same effect as selecting Yes for the Uninstall Superseded Component setting of all components in the patch.*
REG File to Merge at Build	Basic MSI, DIM, InstallScript MSI, Merge Module	If you want to merge a .reg file with the component's registry entries at build time, enter the complete path for the .reg file. As an alternative, you can click the ellipsis button (...) in this setting to browse to the .reg file.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Self-Register	InstallScript, InstallScript Object	Specify whether the component's files are self-registering. (A self-registering file is an OLE server that can place information about itself in the registry so that it is seen by other OLE applications. Such a file can also remove this information from the registry when it is uninstalled.)  ***Important*** • *If you select Yes but one or more files in the component are not self-registering, an error occurs when the installation runs.*  ***Project*** • *InstallScript and InstallScript Object projects support self-registration of .dll files, .exe files, and type libraries—that is, .tlb and .olb files.* *In Basic MSI and InstallScript MSI projects, self-registration is set at the file level, rather than the component level.*
Potentially Locked	InstallScript, InstallScript Object	Specify whether already-installed versions of the component's files may be locked—that is, in use by another product—during installation. If a file is locked during installation or uninstallation, and that file is in a component whose Potentially Locked setting is set to Yes, the file operation is automatically performed after restart. If a file is locked during installation or uninstallation, and that file is in a component whose Potentially Locked setting is set to No, the **OnFileLocked** event handler function is called. (The default code for this function automatically performs the file operation after restart.) Note that InstallShield always treats shared files as potentially locked files.
Compressed	InstallScript, InstallScript Object	Specify whether you want to compress the files in the component if its files are built into a .cab file at build time. This setting has no effect if you select the check box for the component's feature in the Custom Media Layout panel of the Release Wizard.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Overwrite	InstallScript, InstallScript Object	This setting indicates the installation's behavior when it encounters an existing file with the same name as the one being installed; you can selectively replace each file in a component based on its version number or modification date. To change the behavior, click the ellipsis button (...) in this setting. The Overwrite dialog box opens, enabling you to specify the appropriate run-time behavior. ***Tip*** • *If the component consists of files—such as data and configuration files—that end users may have updated, consider selecting the Never Overwrite option.* Files that are associated with a component that has the Never Overwrite option selected for the Overwrite setting are logged for uninstallation; that is, even if such a file is not copied to the target system, because a file with the same name already exists there, that file is removed when the application is uninstalled. To prevent an already existing file from being uninstalled, select Yes for the Shared setting of the file's component.
Difference	InstallScript, InstallScript Object	Specify whether you want InstallShield to include the component's files in a release when you are building a differential release. (When you define a differential release, you specify one or more existing releases to which the current project should be compared when the new differential release is built.) Available options are: • **Include If Appropriate**—InstallShield excludes a file in this component from the differential release if the same file (with the same date, time, size, and attributes) exists in each of the specified comparison releases. This is the default option. • **Include Always**—InstallShield includes all of the component's files in the differential release, even if the same file (with the same date, time, size, and attributes) exists in each of the specified comparison releases.
Link Type	InstallScript, InstallScript Object	Specify whether the component's file links are static or dynamic and, if the latter, how the links are determined at the time of the release build.

Table 12-9 • Component Settings in the General Area (cont.)

Setting	Project Type	Description
Comments	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter comments about this component. Your comments are saved in the project file for your reference and are not used in the installation at any time.

Target Machine

When you select a component in the Components view or the Setup Design view of an InstallScript, InstallScript MSI, or InstallScript Object project, the following settings are available in the Target Machine area:

Table 12-10 • Component Settings in the Target Machine Area

Setting	Project Type	Description
Operating Systems	InstallScript, InstallScript MSI, InstallScript Object	If a component is specific to one or more operating systems, use this setting to indicate those operating systems. If the target machine's operating system does not match one of the operating systems that are specified for this setting, the component is not installed. By default, components are operating system independent, meaning that none of the component's data are specific to certain operating systems. To change the value of this setting, click the ellipsis button (...) in this setting.

Table 12-10 • Component Settings in the Target Machine Area (cont.)

Setting	Project Type	Description
Platform Suite(s)	InstallScript, InstallScript Object	If a component is specific to one or more platform suites, use this setting to indicate the suites. If you specify more than suite, you can indicate whether all or any of the suites must be present on the target machine in order for the component to be installed. By default, components are platform suite independent, meaning that none of the component's data are specific to a particular platform suite.  ***Tip*** • *This setting provides an additional layer of filtering beyond the Operating Systems setting. Select platform suites for the Platform Suite(s) setting only if necessary, and be sure to select only those platform suites that are required for the proper functioning of your application. For example, if a component should be installed on both the Home and Professional editions of Windows XP, you can leave Suite Independent as the value for this setting; selecting Windows XP for the Operating Systems setting encompasses both editions.* You can control the platform suites that your installation supports at run time by calling the **FeatureFilterOS** function. In the **OnFilterComponents** event handler, the framework typically calls this function with the platform suites that match the target system so that only the appropriate components are installed. By calling **FeatureFilterOS**, you can override this default behavior to install or prevent the installation of components based on any platform suite criteria that you specify.

.NET Settings

When you select a component in the Components view or the Setup Design view, the following settings are available in the .NET Settings area:

Table 12-11 • Component Settings in the .NET Settings Area

Setting	Project Type	Description
.NET COM Interop	Basic MSI, DIM, InstallScript MSI, Merge Module	Specify whether you want to use COM interoperation (the ability to call .NET objects from COM) for the component. For example, if you have a Visual Basic .NET class library that defines the ComClass attribute and contains at least one public (COM-callable) function, you can select Yes for this setting; InstallShield extracts the COM Interop information at build time and adds it to the **Registry** table of your .msi database. At run time, registry entries that allow COM objects to call your .NET assembly are created on the target system.  ***Note*** • *This setting is applicable only if the component's key file is a .NET assembly.*
.NET Precompile Assembly	Basic MSI, DIM, InstallScript MSI, Merge Module	Specify whether you want the installation to create a native image from the .NET assembly at run time and install it to the native image cache on the target system. This allows the assembly to load and execute faster, because it restores code and data structures from the native image cache rather than from just-in-time (JIT) compilation.
.NET Assembly	InstallScript, InstallScript Object	Specify whether the component's files should be installed as local .NET assemblies. Available options are: • **Local Assembly**—The component contains a .NET assembly. In addition, the installation performs COM interop registration and configures .NET installer class information for the component at run time. • **Not a .NET Assembly**—The component does not contain a .NET assembly. This is the default option. If you want InstallShield to scan the assembly in this component for .NET dependencies at run time, Local Assembly must be selected for this setting. To learn more, see Identifying Properties and Dependencies of .NET Assemblies.

Table 12-11 • Component Settings in the .NET Settings Area (cont.)

Setting	Project Type	Description
.NET Scan at Build	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	***Project*** *• The functionality for this setting depends on whether the project type is Windows Installer based (Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, or Transform project) or InstallScript based (InstallScript or InstallScript Object project).* • **For a Windows Installer–based project**—Indicate whether you want the key file of this component to be scanned for .NET dependencies, properties, or both at build time. • **For an InstallScript-based project**—Indicate whether you want the files of this component to be scanned for .NET dependencies at build time. For more information, see Identifying Properties and Dependencies of .NET Assemblies.
.NET Application File	Basic MSI, DIM, InstallScript MSI, Merge Module	This setting is used when this component is scanned at build time (based on the .NET Scan at Build setting) or by the Static Scanning Wizard. The scanner uses this setting—along with the component destination—to determine the value of the File Application setting for the assembly. For optimal advertising and repair, select the .NET executable file that uses the assembly in this component. The key file of the component is displayed in this setting if the .NET Scan at Build setting is set to Dependencies and Properties or Properties. You can select any other key or portable executable file in the project. ***Note*** *• This setting is applicable only if the component's key file is a .NET assembly.* *The value for this setting is ignored if the file that is selected is not a .NET assembly file.*
.NET Installer Class	Basic MSI, DIM, InstallScript MSI, Merge Module	***Note*** *• This setting is applicable only if the component's key file is a .NET assembly that contains a class that is derived from System.Configuration.Install.Installer.* If the assembly's Install, Commit, Rollback, and Uninstall methods should be called at the appropriate times at run time, select Yes.

Table 12-11 • Component Settings in the .NET Settings Area (cont.)

Setting	Project Type	Description
.NET Installer Class Arguments	Basic MSI, DIM, InstallScript MSI, Merge Module	To specify arguments that should be passed to the .NET installer class, click the ellipsis button (...) in this setting. The .NET Installer Class Arguments dialog box opens, enabling you to specify arguments for the Install, Commit, Rollback, and Uninstall methods. For sample code, see Reading Properties Passed to the .NET Installer Class.  ***Note*** • *This setting is applicable only if the component's key file is a .NET assembly.*

Run-Time Settings

When you select a component in the Components view or the Setup Design view of an InstallScript or InstallScript Object project, the following settings are available in the Run-Time Settings area:

Table 12-12 • Component Settings in the Run-Time Settings Area

Setting	Project Type	Description
FTP Location	InstallScript, InstallScript Object	If you want to associate an FTP location with the component, enter the FTP address. The location that you enter cannot be accessed from within a script. Note that this same setting is also available for features; however, the location that you specify for the feature setting can be accessed from within a script.
HTTP Location	InstallScript, InstallScript Object	If you want to associate an HTTP address with the component, enter the HTTP address. The location that you enter cannot be accessed from within a script. Note that this same setting is also available for features; however, the location that you specify for the feature setting can be accessed from within a script.
Miscellaneous	InstallScript, InstallScript Object	If you want to associate a text string with the component, enter the text string. The location that you enter cannot be accessed from within a script. Note that this same setting is also available for features; however, the location that you specify for the feature setting can be accessed from within a script.

Advanced Settings for a Component



Project • *The Advanced Settings area for a component is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Advanced Settings area under a component in the Components view (and in the Setup Design view) enables you to fulfill installation requirements for special component types. For example, when you copy an .ocx file to the target system, you must register its classes, ProgIDs, and type libraries so that the file's methods can be properly accessed. Advanced settings use Windows Installer's built-in functionality for registering COM servers; setting up ODBC drivers, data sources, and translators; installing or controlling Windows services; and registering a file association.

By specifying the advanced settings, you can publish your component and register COM servers, file extension servers, and MIME types. If the component is selected, an advanced setting is made on the target system when the component is installed or advertised. That way, the file is ready to execute once it is installed. Publishing components—a type of advertising—is accomplished through the Publishing advanced setting.

Categories of Advanced Settings

The Advanced Settings area under a component in the Components view (and in the Setup Design view) is organized into the following main categories:

Table 12-13 • Categories for the Advanced Settings of a Component

Category	Description
Application Paths	Use this advanced setting to specify the paths to your component's files and folders. When you run your installation, it will create an App Paths key for your component under the following registry key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\ProgramName.exe.
Assembly	Use this advanced setting to install a Win32 or .NET assembly when its component is installed.
COM Registration	In the COM Registration area, you can enter or modify registration information for the COM server that you have set as your component's key file. The COM Registration advanced setting is intended to replace self-registration, which is a violation of Setup Best Practices. The recommended way to specify file registration is to use the Component Wizard and have it automatically extract the necessary information for you. Use the COM Registration explorer only if you need to make advanced modifications.

Table 12-13 • Categories for the Advanced Settings of a Component (cont.)

Category	Description
File Types	This advanced setting registers information about a file type on the target system when the component is installed or advertised.
Services	Use the Services area to install, configure, start, stop, and uninstall Windows services during installation and uninstallation. As an alternative, you can use the Services view to install, configure, start, stop, and uninstall Windows services. If you configure service information in either of these areas, the other area is automatically updated accordingly.
Publishing	Use this advanced setting to publish your component. Publishing is a type of advertising (just-in-time installation) in which no user-interface elements are created for the component during installation, but the component can be installed through the Add or Remove Programs or when an installed component requests the published component from the installer.
Device Driver	The Device Driver advanced setting enables you to specify whether the current component includes a device driver, the type of driver, and the order in which the project's device drivers (not just the current component's device drivers) should be installed.
Other Data	The Other Data advanced setting lists various Windows Installer tables that are related to the component.

Application Path Settings



Project • *The Application Paths area for a component is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The following columns are available in the Application Paths area under the Advanced Settings node of the Components view and the Setup Design view:

Table 12-14 • Assembly Settings

Column	Description
Check Box	To add an application path for the component, select the check box for the executable file whose application path you want to define.

Table 12-14 • Assembly Settings (cont.)

Column	Description
File	This read-only column shows the executable files that are in the selected component.
Application Path	Enter a sequence of directories that the executable file should use as a search path for its DLLs (whether linked to the executable at build time, or loaded with the **LoadLibrary** API). Separate multiple directories with a semicolon (;). Do not use hard-coded paths, such as **C:\MyDir**, for one of your paths. Instead, use Windows Installer folder properties in square brackets, as in **[INSTALLDIR]MyDir** and **[CommonFilesFolder]MyProgram**, or the name of the component after a dollar sign, such as **[$ComponentName]**. The Windows Installer resolves these values and writes the resulting locations to the application path key.

Assembly Settings



Project • *The Assembly area for a component is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The following settings are available in the Assembly area under the Advanced Settings node of the Components view and the Setup Design view:

Table 12-15 • Assembly Settings

Setting	Description
Manifest	For a .NET assembly, this setting is automatically set to the first .exe or DLL file in the component. For a Win32 assembly, this setting is automatically set to the first file in the component with the .manifest extension. The file name of the manifest is the application executable name, followed by the .manifest extension. This setting should be left blank if Global Assembly Cache is selected for the assembly's File Application setting. The Manifest setting cannot be deleted.

Table 12-15 • Assembly Settings (cont.)

Setting	Description
File Application	For a .NET assembly, this setting lets you specify whether to install the assembly in the global assembly cache. Select the appropriate file to install the assembly in your application's private cache, or select Global Assembly Cache to install the assembly in the global assembly cache. For a Win32 assembly, this setting lets you specify whether to install the assembly privately for the selected .exe or to install the assembly to the global assembly cache. If you select Global Assembly Cache, the assembly can be used by any .NET application on the target system. The File Application setting cannot be deleted. The value of this setting affects the component's Destination setting. If the File Application setting is set to Global Assembly Cache, the component's Destination setting is also set to Global Assembly Cache. For optimal advertising and repair, select the executable file that uses the assembly in this component.  ***Note*** • *To install a .NET assembly to the global assembly cache, the assembly must meet certain criteria. For details, consult Microsoft's .NET documentation.* *To install a Win32 assembly to the global assembly cache, your files must be signed and have catalog (.cat) files. For details, see "Creating Signed Files and Catalogs" in the Platform SDK help.*
Properties	Enter the property names and values. To create a new entry, right-click in the grid and click New. For a Win32 assembly, you must enter values for the following properties: type, name, version, language, publicKeyToken, and processorArchitecture. For a private .NET assembly, you must enter values for the following properties: Name, Version, and Culture. For a global .NET assembly, you must enter values for the following properties: Name, Version, Culture, and PublicKeyToken. InstallShield adds the property name and value that you enter to the **MsiAssemblyName** table of your Windows Installer database. You can view the records in this table using the Direct Editor.  ***Note*** • *The property and values that you enter for your assembly must match the information in the assembly's manifest file. If they do not match, the assembly might be left on the target system when your product is uninstalled.*

COM Registration Settings



Project • *The COM Registration area for a component is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The COM Registration area under the Advanced Settings node of the Components view and the Setup Design view is organized into the following categories:

- COM Classes
- ProgIDs
- Type Libraries

COM Classes

When you select a COM class in the COM Registration area of the Components view or the Setup Design view, the following settings are available:

Table 12-16 • COM Class Settings

Setting	Description
ProgID	Select a ProgID for this COM class from the list of ProgIDs created for this COM server. You can also create a ProgID by typing a new name; if you do this, InstallShield adds the new ProgID under the ProgIDs node.

Table 12-16 • COM Class Settings (cont.)

Setting	Description
File Type Mask	Enter the file type mask in the following format: **offset, cb, mask, value** • **offset**—The position, in bytes, from the beginning of the file, indicating where the byte range begins. Number of bytes from the end of the file if the offset value is a negative integer. • **cb**—The number of bytes in the file, starting from the offset value, that will be used to perform a logical AND operation with the mask. • **mask**—A value that is used to perform a logical AND operation with the bits in the file so that the result can be compared with a specified value (the next entry). If you leave this value empty, it is assumed to be all ones. • **value**—The bytes that will be compared with the result of a logical AND operation between the mask and the byte range in a file in order to determine the file's class ID. A file type mask is part of a pattern registered under HKEY_CLASSES_ROOT\FileType*CLSID**pattern*. Windows APIs such as **OleCreateFromFile** and **GetClassFile** can retrieve a class ID for a file by comparing information in this key to the bytes in the file. For example, **0,4,FFFFFFFF,AABBCCDD** would mean that the first four bytes in the file must be AA BB CC DD for the system to use this component's class ID for a specified file. You can specify more than one mask that the file has to match. Separate multiple patterns with a semicolon (;). When the installation creates these keys, the subkeys for each pattern are automatically numbered sequentially starting with the number 0.
Icon File	Enter the fully qualified path to the file that contains the icon associated with this COM class, or click the ellipsis button (...) in this setting to browse to the path. You can specify an .ico, .exe, or .dll file. An icon will be extracted from the file for you if you browse for an executable file or DLL and select a specific icon.
Icon Index	If there is more than one icon resource in the file, enter the index for the appropriate icon. A nonnegative integer refers to the order of the icon resources in the executable file. For example, 0 is for the first icon in the file, 1 for the second, 2 for the third, and so on. Use a negative number to refer to a specific resource ID. For example, the icon index -12 points to the icon with a resource ID of 12.

Table 12-16 • COM Class Settings (cont.)

Setting	Description
AppID	Select an AppID for this DCOM or COM+ class from the list of AppID names that you have already created for this file's registration. An AppID is a GUID that is registered under HKEY_CLASSES_ROOT\AppID used for grouping all of the security and configuration options of distributed COM objects. To create a new AppID, you need to use the Direct Editor: Navigate to the AppID table and enter the information for your new AppID. Once you have created an AppID, you can select it from the list in this setting. To permanently delete an AppID, delete its row in the AppID from within the Direct Editor.
Description	Enter a description for this COM class. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Relative Path	Specify whether you want to register a relative path to the file under its class ID. Available options are: • **No**—The installation registers the fully qualified path to the file under its class ID. Then, when any client application instantiates an object from this class, the system always points to the registered COM server. This is the default option. • **Yes**—The installation registers only the name of the file without any path. Doing so allows you to install side-by-side components, meaning that the operating system uses the copy of the COM server in the current directory and that other applications can maintain separate copies of the file in their respective folders.  ***Note*** • *If the path is relative and the file is not in the system's path, the operating system will not be able to find the class when a client attempts to create an object outside of the server's current directory.*

When you select a LocalServer32, LocalServer, or InprocServer32 context type under a COM class in the COM Registration area of the Components view or the Setup Design view, the following settings are available:

Table 12-17 • COM Class Context Settings

Setting	Description
Default Inproc Handler	Select the InprocHandler value for this LocalServer or LocalServer32 context type.
Argument	Enter the argument that OLE will use when calling this COM server.

ProgIDs

When you select a ProgID in the COM Registration area of the Components view or the Setup Design view, the following settings are available:

Table 12-18 • ProgID Settings

Setting	Description
COM Class	Select the COM class, if any, to which this ProgID refers. If you later change the class ID (CLSID) for the COM class, you will need to select it again from the list to restore the association.
Description	Enter a description of this ProgID. The description is registered as the default value for the ProgID under the registry key HKEY_CLASSES_ROOT*class.object* when this component is installed. When the user right-clicks a file associated with your file extension and then clicks Properties, the General tab displays the description that you enter here. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Icon File	Enter the fully qualified path to the file that contains the icon associated with this ProgID, or click the ellipsis button (...) in this setting to browse to the path. You can specify an .ico, .exe, or .dll file. An icon will be extracted from the file for you if you browse for an executable file or DLL and select a specific icon.

Table 12-18 • ProgID Settings (cont.)

Setting	Description
Icon Index	If there is more than one icon resource in the file, enter the index for the appropriate icon. A nonnegative integer refers to the order of the icon resources in the executable file. For example, 0 is for the first icon in the file, 1 for the second, 2 for the third, and so on. Use a negative number to refer to a specific resource ID. For example, the icon index -12 points to the icon with a resource ID of 12.

When you select a version-independent ProgID in the COM Registration area of the Components view or the Setup Design view, the following settings are available:

Table 12-19 • Version-Independent ProgID Settings

Setting	Description
Description	Enter a description of this version-independent ProgID. The description is registered as the default value for the version-independent ProgID under its root key (HKEY_CLASSES_ROOT*version-independent ProgID*) when this component is installed. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Type Libraries

When you select a type library in the COM Registration area of the Components view or the Setup Design view, the following settings are available:

Table 12-20 • Type Library Settings

Setting	Description
Locale ID	Enter the decimal value of the type library's locale ID. Since type libraries are usually language-neutral, the number zero is the most common language of type libraries.
Version	Provide the version number of the type library. If this setting is left blank, Windows Installer automatically determines the version of the type library at run time. Windows Installer requires the type library version to be in the format *x.y*, where *x* and *y* are decimal (base-10) integers.

Table 12-20 • Type Library Settings (cont.)

Setting	Description
Cost	Enter the cost that is associated with registering this type library. Use non-negative numbers only. Enter the number 1 if no specific value is available.
Help Path	Select the destination folder where the help file for this type library will be installed, or click the ellipsis button (...) to select or create a directory. To use the component's default destination directory, select [INSTALLDIR].
Description	Enter a description of this type library. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

File Type Settings



Project • *The File Types area for a component is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The File Types area under the Advanced Settings node of the Components view and the Setup Design view is organized into the following categories:

- Extension
- Verbs
- MIME Type
- ProgIDs

Extension

When you select an extension in the File Types area of the Components view or the Setup Design view, the following settings are available:

Table 12-21 • Extension Settings

Setting	Description
ProgID	Select the ProgID that should be registered for this file extension. You can also create a ProgID by typing a new name; if you do this, InstallShield adds the new ProgID under the ProgIDs node. A file type's ProgID is an arbitrary string, but it should be unique on the target system. One ProgID naming convention is to append the word *file* to your extension without a dot—the *.ext* extension might use the ProgID *extfile*. Another convention is to name a file-type ProgID after the application used to open the file type, as in **SampleApp.Document**.
MIME Type	Select the MIME type, if any, that is associated with this file extension. To create a new MIME type for your extension, right-click its icon and click New MIME Type.

Verbs

When you select a verb under a file extension in the File Types area of the Components view or the Setup Design view, the following settings are available:

Table 12-22 • Verb Settings

Setting	Description
Command Sequence	Enter a sequence number for the command verb. If this file extension has more than one verb associated with it, the sequence numbers determine the order in which the verbs are displayed on the context menu (also known as a right-click menu or pop-up menu). If you do not specify a sequence number, or if more than one verb has the same sequence number, the verbs are listed alphabetically. The first verb in the command sequence is displayed in bold as the default option on the file type's context menu.

Table 12-22 • Verb Settings (cont.)

Setting	Description
Display Name	Enter the text that you want to display for this verb on the context menu that Windows Explorer displays when an end user right-clicks a file with the current extension. For example, to display **Open with SampleApp** (with an underlined letter *O*) on the context menu for this file extension, enter **&Open with SampleApp**. This setting is optional. If you do not specify a display name, the name of the verb as it appears in the Extensions tree is used on the file-type's context menu on the target system. Note that if you use one of the canonical verbs—such as *open*, *print*, or *find*, and you do not specify a display name, Windows automatically localizes the verb on each system. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Argument	Enter the command-line arguments for this verb. Use **%1** in the argument in place of the selected file name. For example, suppose an end user right-clicks C:\File.ext and **-p %1** is the argument for the verb, the command-line argument becomes **-p C:\File.ext**. In some cases, it may be necessary to place quotation marks around the %1 argument—that is, enter **"%1"** as the argument—to correctly handle file names that contain spaces.

MIME Types

When you select a verb under a file extension in the File Types area of the Components view or the Setup Design view, the following settings are available:

Table 12-23 • Mime Type Settings

Setting	Description
Class ID	Enter the class ID that is associated with this MIME type.

ProgIDs

When you select a ProgID in the File Types area of the Components view or the Setup Design view, several settings are available. For descriptions of each setting, see ProgIDs.

Services Settings



Project • *The Services area for a component is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Note *• A service must be a single executable file (.exe), since the Windows Installer does not support driver services.*

The service executable file must be the key file of its component. For more information, see Component Key Files.

For information about the Services settings, see Services View.

Publishing Settings

Project *• The Publishing area for a component is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You can add the following types of items to the Publishing explorer in the Publishing area under the Advanced Settings node of the Components view and the Setup Design view:

Table 12-24 • Publishing Settings

Setting	Description
ComponentID	The componentID, a GUID, is a category identifier that represents the category of components that are being grouped together as a qualified component. Each componentID must have at least one qualifier. ***Note*** *• Do not confuse the componentID with the GUID that is entered in the Component Code setting for the component; they must be unique values.*
Qualifier	The qualifier is a string that you can use to distinguish this language or version of the component from any other (for example, to specify a language). It must be unique for the component.

The following settings are available when you select a qualifier in the Publishing area:

Table 12-25 • Qualifier Settings

Setting	Description
Application Data	Enter the application data that corresponds with its qualifier. The application data is a text string that may be displayed to end users. This setting is optional. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Device Driver Settings



Project • *The Device Driver area for a component is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The following tabs are available in the Device Driver area under the Advanced Settings node of the Components view and the Setup Design view:

- Common tab
- Sequence tab

Common Tab

The Common tab contains the following run-time settings for device drivers:

Table 12-26 • Device Driver Settings on the Common Tab

Options	Description
Always overwrite any existing device driver	Replace any existing driver for the device with this driver. If this check box is cleared and a driver already exists for the device, your driver is not installed.
Do not show Connect Device to Computer dialog	To prevent the Connect Device to Computer dialog from being displayed during the installation of a device driver if a device matching the driver is not connected to a computer, select this option.

Table 12-26 • Device Driver Settings on the Common Tab (cont.)

Options	Description
Do not create Add or Remove Programs entry for device driver	If you select this check box, the installation does not create an Add or Remove Programs entry for the device driver. If you clear this check box, the installation does create an Add or Remove Programs entry. DIFxApp does not support this feature on Windows Vista and later.
Install unsigned driver files and drivers with missing files	By default, DIFxApp does not install unsigned driver packages or driver packages that have missing files. To override this default behavior, select this check box. Selecting this check box enables you to fully test drivers more easily before shipping final signed versions.
Remove binary files associated with driver during uninstall	By default, when DIFxApp uninstalls a driver package, DIFxApp does not remove the binary files that were copied to the system when it installed the driver. To override this default behavior, select this check box. If you select this check box, DIFxApp removes a binary file from the system only if the binary file is identical to the corresponding binary file in the driver store.  ***Caution*** • *Select the* ***Remove binary files associated with driver during uninstall*** *check box only if you can verify that the driver's binary files are not required by any other driver package or application.*

Table 12-26 • Device Driver Settings on the Common Tab (cont.)

Options	Description
Include all localized installation runtime dialogs	When this check box is selected, the installation uses a custom action runtime .dll that includes dialogs and text for the following languages: • Arabic (Saudi Arabia) • Chinese (People's ROC) • Chinese (Taiwan) • Czech (Czech Republic) • Danish (Denmark) • Dutch (Netherlands) • English (United States) • Finnish (Finland) • French (France) • German (Germany) • Greek (Greece) • Hebrew (Israel) • Hungarian (Hungary) • Italian (Italy) • Japanese (Japan) • Korean (Korea) • Norwegian (Bokmål) (Norway) • Polish (Poland) • Portuguese (Brazil) • Portuguese (Portugal) • Russian (Russia) • Spanish - Modern Sort (Spain) • Swedish (Sweden) • Turkish (Turkey) If you clear this check box, the installation uses runtime dialogs for English only. The English runtime .dll file is smaller than the localized .dll file, so if space is an issue and you do not need the extra language runtime dialogs, clear this check box. This check box applies to all of the components in the project that contain device drivers.

Table 12-26 • Device Driver Settings on the Common Tab (cont.)

Options	Description
Device driver machine architecture	This area has several options. Select the appropriate option. • Device driver targets 32 bit machines • Device driver targets Itanium 64 bit machines • Device driver targets AMD 64 bit machines This setting applies to all of the components in the project that contain device drivers.

Sequence Tab

The Sequence tab enables you to specify the order in which the project's device drivers (not just the current component's device drivers) should be installed.

Other Data Settings



Project • *The Other Data area for a component is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Other Data area for a component is available in the Components view and the Setup Design view.



Task ***Using the Other Data node:***

- Click the Other Data node to view a list of Windows Installer tables that are related to the component. The list also includes the name of any SQL script files associated with a component. In addition, the Other Data node lists the number of records found in the **Complus**, **DuplicateFile**, **Environment**, **IniFile**, **MoveFile**, **RemoveIniFile**, **RemoveRegistry**, and **ReserveCost** tables.
- To go to a table in the Direct Editor, click the table name.

Setup Types View



Project • *The Setup Types view is available in the following project types:*

- *InstallScript*

- *InstallScript MSI*

To create setup types for a Basic MSI project, use the feature's Install Level property.

Setup types enable you to provide different versions of your product to your end users. For example, the default setup types are Complete and Custom. The Complete setup type installs all of the files included in your installation. The Custom setup type lets the end user select which features are installed.

Setup types are based on features. You select the features that you would like to associate with each setup type. Then, when an end user selects a certain setup type, only those features that you associated with that setup type are installed.

By default, each project that you create contains predefined setup types. In the Setup Types view, you can add or remove setup types, rename existing setup types, and change which features are associated with each one.

Default Setup Types

Table 12-27 • Default Setup Types

Setup Type	Project Type	Description
Complete	InstallScript, InstallScript MSI	The Complete setup type includes all of your installation program's features.
Custom	InstallScript, InstallScript MSI	The Custom setup type lets end users select which features they would like to install. Required features should be marked as such, so that they are always installed. However, features such as the online help may not need to be installed. In these cases, the end user can select which of the optional features to install.

Additional Setup Types

You can create additional setup types in this view. To learn how to add setup types to your project, see Working with Setup Types.

Setup Type Settings

When you select a setup type in the Setup Types view, the following settings are available:

Table 12-28 • Setup Type Settings

Setting	Description
Display Name	Enter the name of the setup type. This is the name that is displayed to end users when they run your installation. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 12-28 • Setup Type Settings (cont.)

Setting	Description
Description	Type a description for the setup type. ***Note*** *• This description is displayed to end users during the installation process only if you call SdSetupTypeEx in your script.* When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Comment	Type comments about the setup type. The comments are for your use only and do not appear in the installation. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Packages View

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where appropriate.

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Use the Packages view to add one package to an Advanced UI project, or one or more packages to a Suite/Advanced UI project. For information on how to determine which type of package format to include, see Guidelines for Adding Packages to an Advanced UI or Suite/Advanced UI Project.

This view also lets you import one or more InstallShield prerequisites into your project as .msi packages, .msp packages, or .exe packages, depending on the type of file that is configured to run for the prerequisites.

Use this view to define conditions for each package, associate each package with one or more Advanced UI or Suite/Advanced UI features, and configure other settings for each package.

At run time, the **Setup.exe** file launches each package based on the conditions that you have defined and the order in which you listed the packages in the Packages view.

When you select a package in the Packages view, the following tabs are available:

- Common
- Features

When you select a dynamically linked folder in the Packages view, two settings are available. For more information, see Dynamic Link Settings.

Common Tab



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where appropriate.



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Common tab in the Packages view is organized into the following main categories:

- General
- Operation
- Configuration (This is available for Web Deploy packages.)
- Events (This is available in Suite/Advanced UI projects.)

General Settings

Use a package's General settings to specify details such as the display name and the conditions under which the Advanced UI or Suite/Advanced UI installation should launch the package.

Note that some settings are not applicable to all types of Advanced UI or Suite/Advanced UI package files.

Table 12-29 • General Settings on the Common Tab

Setting	Type of Package File	Description
Package GUID	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Enter a GUID that uniquely identifies this package. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Display Name	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Enter the name that you want to be displayed for this package at run time. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
Product Configuration	Basic MSI project	Select the product configuration that contains the release in the InstallShield project that you want to associate with the selected package. When you start a build of a release for a Suite/Advanced UI project that includes one or more InstallShield project packages, InstallShield first builds the designated releases in the associated InstallShield projects, and includes them as packages in the Suite/Advanced UI installation that it generates.
Release	Basic MSI project, InstallScript project	Select the release in the InstallShield project that you want to associate with the selected package. When you start a build of a release for a Suite/Advanced UI project that includes one or more InstallShield project packages, InstallShield first builds the designated releases in the associated InstallShield projects, and includes them as packages in the Suite/Advanced UI installation that it generates.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Location	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Specify where the package should be located. Available options are: • **Copy From Source Media**—Store the package and its files on the source media. If the Advanced UI or Suite/Advanced UI installation will be run uncompressed from fixed media—such as CD, DVD, or local network—use this option. • **Extract From Setup.exe**—Compress the package and its files into **Setup.exe**, to be extracted at run time, if necessary. If the entire Advanced UI or Suite/Advanced UI installation must be self-contained in **Setup.exe**, select this option. Note that the Download From The Web option results in smaller installations and shorter download time; however, the Extract From Setup.exe option provides for a completely self-contained installation. • **Download From The Web**—Download the package and its files (if necessary) from the URL that is specified for the package. This option is recommended if your installation will be downloaded from the Internet and you want to minimize the Advanced UI or Suite/Advanced UI package size and download time. An Advanced UI or Suite/Advanced UI package is not downloaded if the correct version is already present on the target system. Note that the option that you select in the Location setting may be overridden in the Releases view.
URL	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Enter the URL for the root folder that will contain the selected package and its folders and files. The package and any accompanying files are downloaded from this location to target systems at run time if the package needs to be launched. This setting is applicable if Download From The Web is selected in the Location setting.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Package Type	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	 ***Project*** • *In Suite/Advanced UI projects, this setting is available for edit. In Advanced UI projects, this setting is read-only, since an Advanced UI project has support for only one primary package.* In a Suite/Advanced UI project: Select the type of package that identifies whether the presence of this package on the target system should influence whether the Suite/Advanced UI installation runs in first-time-installation mode or maintenance mode. Available options for this setting are: • **Primary**—The selected package is a main part of the Advanced UI or Suite/Advanced UI installation. At run time, if all of the primary packages in the installation are absent from the target system, the installation runs as a first-time installation. If any of the primary packages are present on the target system, the installation runs in maintenance mode. • **Dependency**—The selected package should not be a factor in determining which mode is used to run the Advanced UI or Suite/Advanced UI installation. For more information, see Primary Packages vs. Dependency Packages in Advanced UI and Suite/Advanced UI Projects.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Cache Path	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Specify where the cached package and other package files should be stored on the end user's system. You can enter a hard-coded value such as **C:\CachedFiles**, but it is recommended that you use a destination property value in the list for the path. The default value is as follows: **[LocalAppDataFolder]Downloaded Installations** The Setup.exe tab in the Releases view lets you specify whether you want to cache the packages on target systems for any packages that are run on target systems and that have a cache path defined. If you are building an uncompressed release, you may want to avoid caching packages on the target system. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
Certificate File	.appx	The versions of Windows that support UWP app packages do not install a sideloading package unless they trust the source of the package. If you are using a custom certificate, such as the one that is generated in Visual Studio, the Suite/Advanced UI installation can make Windows trust the source of your package by adding the custom certificate to the certificate store of target systems. The Suite/Advanced UI installation does this if you include the .cer file in your sideloading package, and specify it here.
Require Elevated Privileges	.msi, .msp, .exe, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Specify whether the package requires elevation on Windows Vista and later and Windows Server 2008 and later systems.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Minor Upgrade Handling	.msi, Basic MSI project	Indicate the appropriate behavior that should occur when an earlier version of the package is present on the target system. Available options are: • **None**—The Advanced UI or Suite/Advanced UI installation launches the package without running it in minor upgrade mode. That is, the Advanced UI or Suite/Advanced UI installation does not set the **REINSTALL** or **REINSTALLMODE** properties for the package. • **Automatic**—The Advanced UI or Suite/Advanced UI installation launches the package in upgrade mode by setting the **REINSTALL** property to *ALL* and the **REINSTALLMODE** property to *vomus*. It does not inform the end user that the product will be upgraded. • **Ask the User**—The Advanced UI or Suite/Advanced UI installation displays a secondary window that asks the end user whether they want to continue. If the end user chooses to continue, the Advanced UI or Suite/Advanced UI installation launches the package in upgrade mode. If the end user chooses to not continue, the package does not run.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Status Messages Reported	.msi, .msp, Basic MSI project	Specify what sort of status messages you want to be available for displaying in the user interface of the Advanced UI or Suite/Advanced UI installation. Available options are: • **Action Text**—The Advanced UI or Suite/Advanced UI installation's status messages include the action text from the standard actions and custom actions in the package. • **Action Text and Action Data**—The Advanced UI or Suite/Advanced UI installation's status messages include the action text as well as action data from the standard actions and custom actions in the package. At run time, the Advanced UI or Suite/Advanced UI installation updates the **ISParcelStatus** property with the action text and, if applicable, the action data in the **ISParcelStatus** property, which is displayed on the InstallationProgress wizard page of the Advanced UI or Suite/Advanced UI installation by default. For example, when the InstallFiles action in an .msi package is executing, the **ISParcelStatus** property is updated with the action text "Copying new files" to inform end users about the current progress of the installation. If you choose to include action data in the status messages, the **ISParcelStatus** property may also be updated with action data such as the name, directory, and size of each file as it is being installed on the target system.
Shared	.msi, .msp, .exe, .appx, InstallScript, Basic MSI project, InstallScript project	Specify whether you want to mark this package as shared. The shared package functionality helps to ensure that if two or more Advanced UI or Suite/Advanced UI installations are sharing a package, the package remains on the target system until all of the Advanced UI and Suite/Advanced UI products are removed. If you select Yes for this setting, ensure that you also use the same package GUID in all Advanced UI and Suite/Advanced UI projects that are sharing the package. To learn more, see Sharing Common Packages Among Different Advanced UI and Suite/Advanced UI Installations.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Enable Logging Support	.msi, .msp, .exe, InstallScript, Basic MSI project, InstallScript project	Specify whether you want the package to generate a log file if the Advanced UI or Suite/Advanced UI installation is launched from the command line with the `/log` command-line parameter. Note that if a package does not have support for logging, it is recommended that you do not enable logging support for that package. If you select Yes, configure each subsetting under this setting as needed. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.
Log File	.msi, .msp, Basic MSI project	This setting is enabled if Yes is selected for the Enable Logging Support setting. Specify a name for the log file. Do not include a path for the file; the Advanced UI or Suite/Advanced UI `/log` command-line parameter lets end users specify the directory for the package log files. If you leave this setting blank, the name of the log file that the installation creates is *`PackageGUID`***.log**, where *`PackageGUID`* is the GUID that is assigned to the package in the Package GUID setting in the Packages view. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.
Log Options	.msi, .msp, Basic MSI project	This setting is enabled if Yes is selected for the Enable Logging Support setting. Specify the Windows Installer log `/L` flags that you want the package to use when generating the log file. For example, to generate a log file that logs everything verbosely, enter the following in this setting: *v For additional valid flags, see the /L description. If you leave this setting blank, the asterisk (*) and v flags are used to generate the log file. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Logging Command Line	.exe, InstallScript, InstallScript project	This setting is enabled if Yes is selected for the Enable Logging Support setting. Specify the command line that the Advanced UI or Suite/Advanced UI installation should pass to the .exe package to enable the logging. Include any appropriate supported flags. If the .exe package that you are configuring supports it, include a path that references the Advanced UI or Suite/Advanced UI property **ISLogDir**, enclosed within square brackets, in place of the path to the directory where the log file should be created. For example, to generate a log file that logs everything verbosely for an .msi package that is run by a **Setup.exe** file that InstallShield built, enter the following command line: `/v"/l*v \"[ISLogDir]FileName.log\""` The Advanced UI or Suite/Advanced UI installation replaces **[ISLogDir]** with the path to the folder that will contain the log file. The Advanced UI or Suite/Advanced UI `/log` command-line parameter lets end users specify the path to the directory for the package log files. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Detection Condition	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	This setting lets you specify one or more conditions that the Advanced UI or Suite/Advanced UI installation should use to evaluate whether the package is already installed on target systems. For example, if the package installs a specific file or registry entry, you can create a condition that specifies the file or registry key that the installation should search for. To add one or more new detection conditions, click the New Condition button in this setting. InstallShield adds a new row under the Detection Condition setting. Select the appropriate option—All, Any, or None—from the list in this row. Then in this row, click the New Condition button, and select the appropriate option to continue building the conditional statement. For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects. If one or more conditional statements are configured, the Detection Condition setting says **(Condition)**. If none are configured, the Detection Condition setting says **(Empty)**.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Eligibility Condition	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	This setting lets you specify one or more conditions that the Advanced UI or Suite/Advanced UI installation should use to determine whether the target system meets the requirements that are necessary for the package to be run. For example, if the package should be run only on 64-bit systems, you could set up an x64 platform requirement in a condition; the Advanced UI or Suite/Advanced UI installation would launch the package only on 64-bit systems. You may also want to set up an eligibility condition to prevent end users from being able to install the current package version over a future newer version. To add one or more new eligibility conditions, click the New Condition button in this setting. InstallShield adds a new row under the Eligibility Condition setting. Select the appropriate option—All, Any, or None—from the list in this row. Then in this row, click the New Condition button, and select the appropriate option to continue building the conditional statement. For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects. If one or more conditional statements are configured, the Eligibility Condition setting says **(Condition)**. If none are configured, the Eligibility Condition setting says **(Empty)**.

Table 12-29 • General Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Windows Features	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	 ***Project*** • *This setting is available in Suite/Advanced UI projects.* This setting lets you specify one or more Windows roles and features that the selected package requires be enabled on target systems that have Windows Vista or later or Windows Server 2008 or later. At run time, if this package is eligible to be installed and it requires one or more Windows roles and features that are disabled, the Suite/Advanced UI installation enables them before launching the package. To specify a required Windows role or feature, click the Add New Windows Feature button in this setting, and then click one of the available options. The Custom option lets you specify any Windows role or feature that is not listed as an option in this setting. When you select an option, InstallShield adds a new Windows Feature row under the Windows Features setting. Configure the Windows Feature setting as needed. For more information, see Enabling Windows Roles and Features During a Suite/Advanced UI Installation.
Release Flags	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	If you want to use release flags to be able to selectively include and exclude this package from different builds of your Advanced UI or Suite/Advanced UI installation, enter one or more release flags to identify this package. Separate multiple flags with a comma.

Operation Settings

Use a package's Operation settings to specify information such as the behavior that should occur if the Advanced UI or Suite/Advanced UI installation is installing, removing, repairing, or modifying the package. This includes information such as the command line that should be used when launching the package and the behavior that should occur if the package requires that the target system be rebooted.

Note that some settings are not applicable to all types of Advanced UI and Suite/Advanced UI package formats.

Table 12-30 • Operation Settings on the Common Tab

Setting	Type of Package File	Description
Install	.msi, .msp, .exe, .appx, InstallScript, Basic MSI project, InstallScript project	If the Advanced UI or Suite/Advanced UI installation runs in first-time installation mode, specify whether you want this package to be launched as a first-time installation. The install operation is applicable if the target system meets the package's eligibility conditions and if this package's product is not already installed. If you select Yes for this setting, configure the subsettings under this setting as needed. If you select No, the subsettings under this setting are disabled.
Remove	.msi, .exe, .appx, InstallScript, Basic MSI project, InstallScript project	If the Advanced UI or Suite/Advanced UI installation runs in remove mode, specify whether you want the product that this package installed to be removed. The remove operation is applicable if the target system meets the package's eligibility conditions and if this package's product is installed. If you select Yes for this setting, configure the subsettings under this setting as needed. If you select No, the subsettings under this setting are disabled.
Repair	.msi, .exe, InstallScript, Basic MSI project, InstallScript project	If the Advanced UI or Suite/Advanced UI installation runs in repair mode, specify whether you want this package to be launched in repair mode to repair the product that is installed by this package. The repair operation is applicable if the target system meets the package's eligibility conditions and if this package's product is installed. If you select Yes for this setting, configure the subsettings under this setting as needed. If you select No, the subsettings under this setting are disabled.
Modify	.msi, .msp, .exe, InstallScript, Basic MSI project, InstallScript project	If the Advanced UI or Suite/Advanced UI installation runs in modify mode, specify whether you want this package to be launched in maintenance mode to enable end users to install, remove, or modify the product that is installed by this package. The modify operation is applicable if the target system meets the package's eligibility conditions and if this package's product is installed. If you select Yes for this setting, configure the subsettings under this setting as needed. If you select No, the subsettings under this setting are disabled.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Target	.msi, .msp, .exe, InstallScript	This setting is a subsetting under the Install setting for .msi, .msp, .exe, and InstallScript packages. It is also under the Remove, Repair, and Modify settings for .exe packages. Select the file in the package—the .msi file, the .msp file, or the executable file setup launcher—that the Advanced UI or Suite/Advanced UI installation should invoke.
EXE Command Line	.exe	This setting is a subsetting under the Install, Remove, Repair, and Modify settings. If applicable, specify the command line that should be used to launch the target file (that is, the file that is selected in the Target setting) whenever the Advanced UI or Suite/Advanced UI installation is running with a user interface. Do not include the name of the file in this setting. If all of the customization can be done without end-user intervention, consider including the command-line parameter for running the .exe package silently. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. For more information, including valid types of entries for this setting, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
MSI Command Line	.msi, Basic MSI project	This setting is a subsetting under the Install, Remove, Repair, and Modify settings. If applicable, specify the command line that should be used to launch the target file (that is, the file that is selected in the Target setting) whenever the Advanced UI or Suite/Advanced UI installation is running with a user interface. Do not include the name of the file in this setting. If you are configuring the Install and Remove operations for an .msi package, the only type of command-line parameters that you should enter are Windows Installer properties. If you are configuring the Repair and Modify operations for an .msi package, the only type of command-line parameters that you should enter are Windows Installer feature properties. To enter a Windows Installer property, use the following format: `MYPROPERTYNAME=MyPropertyValue` Note that an Advanced UI or Suite/Advanced UI installation always launches .msi packages silently. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the .msi package. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. For more information, including valid types of entries for this setting, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description

MSP Command Line	.msp	This setting is a subsetting under the Install, Remove, Repair, and Modify settings.
		
		If applicable, specify the command line that should be used to launch the target file (that is, the file that is selected in the Target setting) whenever the Advanced UI or Suite/Advanced UI installation is running with a user interface. Do not include the name of the file in this setting.
		
		To update all of the features through an .msp package, you would enter command-line properties such as the following:
		
		`REINSTALLMODE=vomus REINSTALL=ALL`
		
		To update only certain features that are included in the .msp package, you would set **REINSTALL** to a comma-separated list of features that you want to be updated:
		
		`REINSTALLMODE=vomus`
		`  REINSTALL=Feature1,Feature3,Feature5`
		
		Note that an Advanced UI or Suite/Advanced UI installation always launches .msp packages silently. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the .msp package.
		
		When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
		
		For more information, including valid types of entries for this setting, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description

Command Line	InstallScript, InstallScript project	This setting is a subsetting under the Install, Remove, Repair, and Modify settings.
		
		If applicable, specify the command line that should be used to launch the InstallScript package that is associated with the selected target file (that is, the **data1.hdr** file that is selected in the Target setting) whenever the Advanced UI or Suite/Advanced UI installation is running with a user interface. Do not include the name of the file in this setting.
		
		If you are configuring the Install and Modify operations for an InstallScript package, you can use the Advanced UI or Suite/Advanced UI properties **ISFeatureInstall** and **ISFeatureRemove** in your command line. Set these properties to a comma-delimited list of feature names as follows:
		
		`ISFeatureInstall=Feature1,Feature2`
		`ISFeatureRemove=Feature3`
		
		In the above example, **Feature1** and **Feature2** are selected to be installed; **Feature3** is selected to be removed, if it is present.
		
		Note that an Advanced UI or Suite/Advanced UI installation launches InstallScript packages silently by default. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the InstallScript package.
		
		When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
		
		For more information, including valid types of entries for this setting, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
EXE Silent Command Line	.exe	This setting is a subsetting under the Install, Remove, Repair, and Modify settings. If applicable, specify the command line that should be used to launch the target file (that is, the file that is selected in the Target setting) whenever the Advanced UI or Suite/Advanced UI installation is running silently (that is, without a user interface). Do not include the name of the file in this setting. If all of the customization can be done without end-user intervention, consider including the command-line parameter for running the .exe package silently. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. For more information, including valid types of entries for this setting, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
MSI Silent Command Line	.msi, Basic MSI project	This setting is a subsetting under the Install, Remove, Repair, and Modify settings. If applicable, specify the command line that should be used to launch the target file (that is, the file that is selected in the Target setting) whenever the Advanced UI or Suite/Advanced UI installation is running silently (that is, without a user interface). Do not include the name of the file in this setting. If you are configuring the Install and Remove operations for an .msi package, the only type of command-line parameters that you should enter are Windows Installer properties. If you are configuring the Repair and Modify operations for an .msi package, the only type of command-line parameters that you should enter are Windows Installer feature properties. To enter a Windows Installer property, use the following format: `MYPROPERTYNAME=MyPropertyValue` Note that an Advanced UI or Suite/Advanced UI installation always launches .msi packages silently. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the .msi package. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. For more information, including valid types of entries for this setting, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description

MSP Silent Command Line	.msp	This setting is a subsetting under the Install, Remove, Repair, and Modify settings.
		
		If applicable, specify the command line that should be used to launch the target file (that is, the file that is selected in the Target setting) whenever the Advanced UI or Suite/Advanced UI installation is running silently (that is, without a user interface). Do not include the name of the file in this setting.
		
		To update all of the features through an .msp package, you would enter command-line properties such as the following:
		
		`REINSTALLMODE=vomus REINSTALL=ALL`
		
		To update only certain features that are included in the .msp package, you would set **REINSTALL** to a comma-separated list of features that you want to be updated:
		
		`REINSTALLMODE=vomus`
		`  REINSTALL=Feature1,Feature3,Feature5`
		
		Note that an Advanced UI or Suite/Advanced UI installation always launches .msp packages silently. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the .msp package.
		
		When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
		
		For more information, including valid types of entries for this setting, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description

Silent Command Line	InstallScript, InstallScript project	If applicable, specify the command line that should be used to launch the InstallScript package that is associated with the selected target file (that is, the **data1.hdr** file that is selected in the Target setting) whenever the Advanced UI or Suite/Advanced UI installation is running silently (that is, without a user interface). Do not include the name of the file in this setting.
		
		If you are configuring the Install and Modify operations for an InstallScript package, you can use the Advanced UI or Suite/Advanced UI properties **ISFeatureInstall** and **ISFeatureRemove** in your command line. Set these properties to a comma-delimited list of feature names as follows:
		
		`ISFeatureInstall=Feature1,Feature2`
		`ISFeatureRemove=Feature3`
		
		In the above example, **Feature1** and **Feature2** are selected to be installed; **Feature3** is selected to be removed, if it is present.
		
		Note that an Advanced UI or Suite/Advanced UI installation launches InstallScript packages silently by default. Therefore, it is not necessary to pass a command-line parameter that hides the user interface of the InstallScript package.
		
		When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
		
		For more information, including valid types of entries for this setting, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Exit Behavior	.exe	This setting is a subsetting under the Install, Remove, Repair, and Modify settings. If a detection condition is still not met on a target system after the package has been run, either one of the following may be true: • The executable file package was not run successfully. • The detection conditions that were specified for the package were not accurate. For example, a detection condition may indicate that the executable file needs to be launched if a particular file does not exist on the target system. If the file is still missing even after the Advanced UI or Suite/Advanced UI installation has run the executable file package, it is possible that the condition was created erroneously. Specify the behavior that should occur if one or more of the detection conditions still indicate that the package needs to be launched. Available options are: • **Ask whether to continue the setup**—If the Advanced UI or Suite/Advanced UI installation should display a message box that prompts the end user to specify whether the Advanced UI or Suite/Advanced UI installation should continue, select this option. • **Abort the setup**—If the Advanced UI or Suite/Advanced UI installation should end without proceeding further, select this option. • **Continue the setup**—If the Advanced UI or Suite/Advanced UI installation should essentially ignore the executable file's unmet condition and proceed to the next package in the installation (if one is required), select this option.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Reboot Request	.msi, .msp, .exe, InstallScript, Basic MSI project, InstallScript project	This setting is a subsetting under the Install, Remove, Repair, and Modify settings. The installation of an Advanced UI or Suite/Advanced UI package may require that the target system be restarted, as described in Restarting a Target System for an Advanced UI or Suite/Advanced UI Package. Specify the behavior that should occur if the package requires that a target system be restarted. Available options are: • **Allow the machine to reboot**—If the need for a restart is detected, the Advanced UI or Suite/Advanced UI installation exits and lets the package trigger a restart. If the package restarts the target system, the Advanced UI or Suite/Advanced UI installation continues with the next package in the Advanced UI or Suite/Advanced UI after the restart. If the package does not trigger a restart, the Advanced UI or Suite/Advanced UI installation resumes after the next restart. • **Ignore the reboot request**—The Advanced UI or Suite/Advanced UI installation continues without restarting the target system. Thus, if it appears that a restart is required but you want the Advanced UI or Suite/Advanced UI installation to try to skip it, select this option. If the package restarts the target system, the Advanced UI or Suite/Advanced UI installation then resumes with the next package in the installation.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Reboot Request (cont.)		• **Delay the prompt, then exit or reboot the machine**—If the package requires that the target system be restarted, the restart is postponed until a subsequent package triggers a restart, or until the end of the Advanced UI or Suite/Advanced UI installation, after the last package has completed. If the restart occurs because of a subsequent package, the restart behavior follows the reboot behavior that is configured for that subsequent package. If the restart occurs at the end of the Advanced UI or Suite/Advanced UI installation, the Advanced UI or Suite/Advanced UI installation displays a message box that asks the end user whether to restart. If the end user chooses not to allow the restart, the Advanced UI or Suite/Advanced UI installation exits instead of restarting the target system. • **Prompt, then exit or reboot the machine**—If the Advanced UI or Suite/Advanced UI installation detects that the package requires that the target system be restarted, the Advanced UI or Suite/Advanced UI installation displays a message box that asks the end user whether to restart. If the end user chooses to allow the restart, the Advanced UI or Suite/Advanced UI installation resumes after the restart. If the end user chooses not to allow the restart, the Advanced UI or Suite/Advanced UI installation exits instead of restarting the target system.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Reboot Request (cont.)		• **Always prompt, then exit or reboot the machine**—When the package installation is complete, the Advanced UI or Suite/Advanced UI installation displays a message box that asks the end user whether to proceed with the restart, even if the Advanced UI or Suite/Advanced UI installation has not detected the need for a restart. If the end user chooses to allow the restart, the Advanced UI or Suite/Advanced UI installation resumes after the restart. If the end user chooses not to allow the restart, the Advanced UI or Suite/Advanced UI installation exits instead of restarting the target system. • **Always reboot the machine**—The Advanced UI or Suite/Advanced UI installation restarts the target system regardless of whether the Advanced UI or Suite/Advanced UI installation detects the need for a restart. The Advanced UI or Suite/Advanced UI installation resumes after the restart. For details on determining the best option to select, see Specifying the Behavior for an Advanced UI or Suite/Advanced UI Package that Requires a Restart.
Reboot Codes	.exe	This setting is a subsetting under the Install, Remove, Repair, and Modify settings. If the selected package requires that the target system be restarted after the product is installed, type the return code in this setting. If multiple return codes exist, list each one separated by a comma. If you do not know the return codes for the file that you are launching as the Advanced UI or Suite/Advanced UI package, contact the author of the package. For more information on Advanced UI or Suite/Advanced UI packages that require a restart, see Restarting a Target System for an Advanced UI or Suite/Advanced UI Package.

Table 12-30 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Force Application Shutdown	.appx	If the selected sideloading app package (or any package that depends on this package) is in use at installation run time, you can require that the processes that are associated with the package are forced to shut down so that registration can proceed. Use this setting to indicate whether you want to force application shutdown in that scenario.

Configuration Settings

Use a Web Deploy package's Configuration settings to specify the values of various Web Deploy settings. For more information, see Adding a Web Deploy Package to a Suite/Advanced UI Project.

Table 12-31 • Operation Settings on the Common Tab

Setting	Type of Package File	Description
Computer Name	Web Deploy	Specify the name of the server where the Web Deploy package should be deployed. You can leave the Suite/Advanced UI property that InstallShield assigns to this setting by default when you add the package to your project, or you can specify your own property (enclosed within square brackets). As an alternative, you can specify a hard-coded value. For more information, see Adding a Web Deploy Package to a Suite/Advanced UI Project.
User Name	Web Deploy	Specify the user name of the end user who will be deploying the Web Deploy package. You can leave the Suite/Advanced UI property that InstallShield assigns to this setting by default when you add the package to your project, or you can specify your own property (enclosed within square brackets). As an alternative, you can specify a hard-coded value. For more information, see Adding a Web Deploy Package to a Suite/Advanced UI Project.

Table 12-31 • Operation Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Password	Web Deploy	Specify the password for the user account that will be used to deploy the Web Deploy package. You can leave the Suite/Advanced UI property that InstallShield assigns to this setting by default when you add the package to your project, or you can specify your own property (enclosed within square brackets). As an alternative, you can specify a hard-coded value. For more information, see Adding a Web Deploy Package to a Suite/Advanced UI Project.
Site	Web Deploy	Specify the site where the Web Deploy package should be deployed. You can leave the Suite/Advanced UI property that InstallShield assigns to this setting by default when you add the package to your project, or you can specify your own property (enclosed within square brackets). As an alternative, you can specify a hard-coded value. For more information, see Adding a Web Deploy Package to a Suite/Advanced UI Project.
Parameters	Web Deploy	If the parameters XML file for the Web Deploy package contains various configurable parameters, the Parameters subsettings let you specify values for those parameters. You can leave the Suite/Advanced UI properties that InstallShield assigns to these subsettings by default when you add the package to your project, or you can specify your own properties (enclosed within square brackets). As an alternative, you can specify a hard-coded value. For more information, see Adding a Web Deploy Package to a Suite/Advanced UI Project.

Events Settings



Project • *The Events area is available in Suite/Advanced UI projects.*

Use a package's Events settings to schedule run-time actions that you want the Suite/Advanced UI installation to launch when installing, removing, repairing, or modifying the package. For more information, see Using Actions to Extend the Behavior of a Suite/Advanced UI Installation.

Table 12-32 • Events Settings on the Common Tab

Setting	Type of Package File	Description
Package Configuring	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	This setting lets you specify one or more run-time actions that the Suite/Advanced UI installation should launch before installing, removing, repairing, or modifying the package. The actions must already be added under the Actions explorer in the Events view. To specify an action, click the Add Events button in this setting, and then click the action. InstallShield adds a new Action setting under the Package Configuring setting, and sets its value to the name of the action that you selected. Use the buttons in the Action setting to specify one or more conditions that the Suite/Advanced UI installation should use to evaluate whether the action should be launched.
Package Configured	.msi, .msp, .exe, .appx, InstallScript, Basic MSI project, InstallScript project	This setting lets you specify one or more run-time actions that the Suite/Advanced UI installation should launch after installing, removing, repairing, or modifying the package. The actions must already be added under the Actions explorer in the Events view. To specify an action, click the Add Events button in this setting, and then click the action. InstallShield adds a new Action setting under the Package Configuring setting, and sets its value to the name of the action that you selected. Use the buttons in the Action setting to specify one or more conditions that the Suite/Advanced UI installation should use to evaluate whether the action should be launched.
Action	.msi, .msp, .exe, .appx, InstallScript, Basic MSI project, InstallScript project	This read-only setting shows the name of an action that is associated with the selected package and defined in the Events view. To define a condition that determines whether this action is run for this package, configure the subsettings under the Action setting as needed. If you associate more than one action with the package, list the actions from top to bottom in chronological order, according to when they are launched at run time. To change the order of the actions, point to the Move Condition button in the setting of an action that you want to move, and the click the appropriate direction (Move Up or Move Down).

Table 12-32 • Events Settings on the Common Tab (cont.)

Setting	Type of Package File	Description
Condition	.msi, .msp, .exe, .appx, InstallScript, Basic MSI project, InstallScript project	This setting lets you specify one or more conditions that the Suite/Advanced UI installation should use to evaluate whether the installation should run an action for the selected package. To add one or more new action conditions, click the New Condition button in this setting. InstallShield adds a new row under the Condition setting. Select the appropriate option—All, Any, or None—from the list in this row. Then in this row, click the New Condition button, and select the appropriate option to continue building the conditional statement. For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects. If one or more conditional statements are configured, the Condition setting says **(Condition)**. If none are configured, the Condition setting says **(Empty)**.
Display Text	.msi, .msp, .exe, .appx, InstallScript, Basic MSI project, InstallScript project	Enter text that describes the selected action. For example, if the action is running a script that configures the target system, you could enter the following string: `Configuring the system` If the installation launches the action when the InstallationProgress wizard page is displayed, the text that you enter is displayed on the wizard page. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. When you type a value for this setting, you can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

Features Tab



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where appropriate.



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Features tab in the Packages view displays the feature tree for the Advanced UI or Suite/Advanced UI project. Use this tab to select the check box of each feature that you want to contain the selected package. Clear the check box of each feature that should not contain the selected package.

Dynamic Link Settings



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where appropriate.



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Dynamic Link pane in the Packages view shows settings for the dynamic link that has been created for a package in the project. The Source and Filter settings are available for all dynamic links. The other settings may be available as subsettings under the Filter setting.

Table 12-33 • Dynamic Link Settings

Setting	Description
Source	This setting shows the path to the source folder that contains the package that you want to include in your Advanced UI or Suite/Advanced UI project. It also is the source of dynamically included files for the package. Instead of hard-coding a location, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.

Table 12-33 • Dynamic Link Settings (cont.)

Setting	Description
Filter	This setting lets you specify one or more filters that you want to configure for including and excluding files and folders from the project. To add a filter, click the **New filter** button in this setting. InstallShield adds a new row that you can use to define the filter. To remove a filter, click the subsetting of the filter that you want to remove, and in that subsetting, click the **Delete this filter** button. To learn more, see Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project.
Include File	Specify the name of the dynamic file that you want to include in your dynamic link. You can use an asterisk (*) as a wild-card character. You can also include a path that is relative to refer to the source folder. For example, to refer to a **ReadMe.txt** file in a subfolder called **MyDirectory**, where **MyDirectory** is a subfolder of the folder that contains the package, you could use the following filter: **.\MyDirectory\ReadMe.txt** For more information, including background on how InstallShield determines which files and folders to include in and exclude from the release, see Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project.
Exclude File	Specify the name of the dynamic file that you want to exclude from your dynamic link. You can use an asterisk (*) as a wild-card character. You can also include a path that is relative to refer to the source folder. For example, to refer to a **ReadMe.txt** file in a subfolder called **MyDirectory**, where **MyDirectory** is a subfolder of the folder that contains the package, you could use the following filter: **.\MyDirectory\ReadMe.tx**t For more information, including background on how InstallShield determines which files and folders to include in and exclude from the release, see Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project.

Table 12-33 • Dynamic Link Settings (cont.)

Setting	Description
Include Folder	Specify the name of the dynamic folder whose files you want to include in your dynamic link. You can use an asterisk (*) as a wild-card character. You can also include a path that is relative to refer to the source folder. For example, to refer to a **MyLibrary**, where **MyLibrary** is a subfolder of the folder that contains the package, you could use the following filter: **.\MyLibrary** For more information, including background on how InstallShield determines which files and folders to include in and exclude from the release, see Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project.
Exclude Folder	Specify the name of the dynamic folder whose files you want to exclude in your dynamic link. You can use an asterisk (*) as a wild-card character. You can also include a path that is relative to refer to the source folder. For example, to refer to a **MyLibrary** folder, where **MyLibrary** is a subfolder of the folder that contains the package, you could use the following filter: **.\MyLibrary** For more information, including background on how InstallShield determines which files and folders to include in and exclude from the release, see Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project.

DIM References View



Project • *The DIM References view is available in Basic MSI projects.*

Use the DIM References view to add and manage references to DIM projects in your Basic MSI project. This view also enables you to view any pertinent instructions that the DIM author noted, associate the DIM references with features in the Basic MSI project, override the default destination of the files in the DIM, schedule custom actions, and more.

The DIM support in InstallShield is targeted toward engineering teams who want to foster collaboration, enable distributed installation development, and maximize efficiency in their organizations. A DIM is a feature-sized collection of related items such as product files, shortcuts, registry entries, text file changes, IIS Web sites, and other elements that together make up a discrete, logically separated portion of a product installation. Once you have created a DIM, you can include references to it in one or more Basic MSI installation projects.

When you select a DIM reference in the DIM References view, several tabs are available:

- General tab
- Instructions tab
- Build Options tab
- Features tab

- Sequences tab

General Tab



Project • *The DIM References view is available in Basic MSI projects.*

The General tab in the DIM References view displays the following read-only settings for the selected DIM reference. Note that the values in these settings are not displayed or used at run time. You can use these settings as a reference for internally identifying a project.

Table 12-34 • Settings on the General Tab in the DIM References View

Setting	Description
File	This setting indicates the name of the selected DIM project file (.dim).
Name	This setting indicates the name of the feature or unit that this DIM represents. This setting is configured in the Subject setting in the General Information view of the DIM project.
DIM GUID	This setting displays the GUID that uniquely identifies the DIM. Whenever you create a new DIM, InstallShield generates a new GUID for it.
Package Code	This setting identifies the package code, which is the GUID for your DIM.
Version	This setting identifies the version number of the DIM.
Description	This setting indicates a description of the DIM. This setting is configured in the Title setting in the General Information view of the DIM project.
Author	This setting is the name of the author of this DIM.
Link to	This setting indicates the location of the selected DIM project file (.dim).

Instructions Tab



Project • *The DIM References view is available in Basic MSI projects.*

The Instructions tab in the DIM References view displays the guidance, tips, or other comments that the author of the DIM project entered in the Build Instructions setting of the General Information view. These instructions may contain critical information that is necessary to properly configure any Basic MSI project that contains a references to this DIM.

Build Options Tab



Project • *The DIM References view is available in Basic MSI projects.*

The Build Options tab in the DIM References view displays the following read-only settings for the selected DIM reference:

Table 12-35 • Settings on the Build Options Tab in the DIM References View

Setting	Description
Destination	Select the folder on the target system into which the DIM's files should be installed, or click the ellipsis button (...) to select or create a directory. This setting lets you override the destination that was set in the DIM project. The default value of this setting is the value that was set in the DIM project. To install the files to the product's default destination folder, select **[INSTALLDIR]**. If you want the destination to be configurable at run time, the destination folder that you specify must be a public property (containing all uppercase letters). You can also set the destination folder of a component's feature. For information on the effects of setting the destination for a feature as well as its components, see Component Destination vs. Feature Destination.
Conflict Resolution	Indicate how you want to resolve conflicts between values of settings in the Basic MSI project with those in the selected DIM project. Available options are: • **Use Base Project Value**—If InstallShield encounters a conflict when merging the DIM data into the Basic MSI installation at build time, InstallShield uses the value that is in the Basic MSI project instead of the value that is in the DIM project. • **Use DIM Project Value**—If InstallShield encounters a conflict when merging the DIM data into the Basic MSI installation at build time, InstallShield uses the value that is in the DIM project instead of the value that is in the Basic MSI project. For more information, see Resolving Build-Time Conflicts Between a Basic MSI Project and a DIM Reference.

Features Tab



Project • *The DIM References view is available in Basic MSI projects.*

The Features tab in the DIM References view displays the feature tree for the Basic MSI project. Use this tab to select the check box of each feature that you want to contain the selected DIM. Clear the check box of each feature that should not contain the selected DIM.

Sequences Tab



***Project** • The DIM References view is available in Basic MSI projects.*

Sequences direct all of the actions that are performed during the installation process—from file transfer to user interface display. These actions are given a number in the sequence, which then executes from the lowest number to the highest. Use the Sequences tab in the DIM References view to insert DIM actions and dialogs into a sequence, or edit the sequence timeline. To learn more, see Scheduling Custom Actions and Dialogs from a DIM Reference.

Application Data View



***Project** • The Application Data view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Application data includes all of the files that you add to your project.

Files and Folders



***Project** • The Files and Folders view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Files and Folders view lets you add files to your project. This view behaves similarly to Windows Explorer, enabling you to drag and drop files into your project. For installation projects, you can select which feature and component you would like your files associated with and the destination directory for those files.

Redistributables



Project *• The Redistributables view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Redistributables view enables you to add InstallShield objects and merge modules to Windows Installer–based projects. Including redistributables to your project lets you add distinct pieces of functionality to your installation. Examples of these types of functionality include Visual Basic run-time .dll files and the Microsoft C run-time libraries.

You can also use the Redistributables view to add InstallShield prerequisites to Basic MSI and InstallScript MSI projects. An InstallShield prerequisite is an installation for a product or technology framework that is required by your product. You can add any of the existing InstallShield prerequisites to your installation projects and configure many of their settings. You can also create your own InstallShield prerequisites, and add them to your projects.

Including InstallShield prerequisites in your project enables you to chain multiple installations together, bypassing the Windows Installer limitation that permits only one Execute sequence to be run at a time. The **Setup.exe** setup launcher serves as a bootstrap application that manages the chaining.

Prerequisites



Project *• The Prerequisites view is available in InstallScript projects.*

The Prerequisites view enables you to add InstallShield prerequisites to InstallScript projects.

Objects



Project *• The Objects view is available in the following project types:*

- *InstallScript*
- *InstallScript Object*

The Objects view enables you to add objects and merge modules to InstallScript and InstallScript Object projects.

Files and Folders View



Project *• The Files and Folders view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



Note *• The Files and Folders view from within Microsoft Visual Studio differs from the Files and Folders view in InstallShield. For more information, refer to Adding References to Visual Studio Solutions.*

The main purpose of most installations is to transfer files from the source medium to the target destination. With InstallShield, adding files to your project is a simple drag-and-drop process in the Files and Folders view. This view contains a files explorer that enables you to drag and drop files from folders on your source machine to folders on the destination machine. The two left panes in the files explorer contain folders, and the two right panes display the files that are located within those folders.



Project *• In installation projects (including InstallScript and Basic MSI), the Files and Folders view includes the View Filter list above the files explorer panes. This list contains all of the features in your project. You can use this filter to show and hide files and folders in the destination panes of this view:*

- *To see in this view only files and folders that belong to a particular feature, select that feature in the View Filter list.*
- *To add a file or folder to a specific feature, select that feature in the View Filter list. Then add the file or folder to the appropriate location in the* ***Destination computer's folders*** *pane.*
- *To see all of the files and folders that are in your project, select the All Application Data option in the View Filter list.*

The following four panes are available in the Files and Folders view.

Table 12-36 • Panes in the Files and Folders View

Pane	Description
Source computer's folders (Top Left)	The **Source computer's folders** pane is similar to the left pane in Windows Explorer. This pane contains folders located either locally or on a network. From here, you can navigate to the folder that contains the files that you want to add to your installation.
Source computer's files (Top Right)	The **Source computer's files** pane displays the files contained in the currently selected folder of the Source computer's folders pane. You can drag files from this pane to a destination folder in the Destination computer's pane. The files you drop into the bottom panes are added to your installation project.
Destination computer's folders (Bottom Left)	In the **Destination computer's folders** pane, you can add files to destination folders or components, create new destination folders, or modify existing components.
Destination computer's files (Bottom Right)	The **Destination computer's files** pane displays all the files you have added to the currently selected destination folder. By right-clicking a file in this pane, you can copy, paste, or delete the file, or edit the file's properties. For Windows Installer-based projects only, you can also right-click a file to set or clear it as the key file of its component.

You can add files to your installation project in one of three different ways:

Table 12-37 • Methods for Adding Files to a Project

Method	Description
Dragging and Dropping Files	You can add files to your project using the Files explorer in the Files and Folders view. The top two panes in this view are functionally equivalent to Windows Explorer. The bottom two panes represent the destination for your files. You can drag source files from the top pane to the destination folder in the bottom pane. InstallShield also provides a context menu with additional drag and drop options that allow you to drag folder only and to preserve the source structure.
Dynamic File Linking	The second way to add files to your project is by linking to the contents of an entire folder, or to specific files in the folder. This method allows you to point to a specific folder, either locally or on a network, that contains files for your installation. Every time you build your installation, the contents of the folder are added to your feature. Additionally, you can use wildcards to filter which files are added to your project. For more information, see Creating a Dynamic Link.  ***Note*** • *To improve performance, the list displayed in the Files and Folders view is truncated when there are more than a predefined number of files in a dynamically linked file with subfolders. **List Truncated** appears as the first item in the file list when this occurs. All files contained in the subfolders are built into your project.*
Dependency Scanning	The final way to add files to your project is accomplished through the Dependency Scanners view. This view contains three wizards that can scan your project, a running application, or a Visual Basic project for possible dependency files and add them to your project. All three of these wizards can also be launched from the Project menu.

Destination Folders



Project • *Some folders are not available in all project types. Project-specific differences are noted where appropriate.*

When you add files to your installation project, you do so by placing them in a destination folder. The following destination folders are provided by default. Each one is dynamic, meaning that they do not use hard-coded paths. Instead, the value for each destination folder is obtained from the operating system of the target machine.

Folders in Basic MSI and InstallScript MSI Projects

The following folders are available only for Basic MSI and InstallScript MSI projects:

Table 12-38 • Default Destination Folders for Basic MSI and InstallScript MSI Projects

Folder Name	Description
AdminToolsFolder	Points to the folder where administrative tools are located.

Table 12-38 • Default Destination Folders for Basic MSI and InstallScript MSI Projects (cont.)

Folder Name	Description
AppDataFolder	This property holds the full path to the current user's Application Data folder. A typical value of this property is: **C:\Users***UserName***\AppData\Roaming**
CommonAppDataFolder	This is the full path to the folder containing application data for all users. A common path is: **C:\ProgramData**
CommonFiles64Folder	The value of this property is the full path to the 64-bit Common Files folder. This property requires Windows Installer version 2.0 or later.
CommonFilesFolder	The value of this property is the full path to the 32-bit Common Files folder.
DesktopFolder	This property is used to hold the full path to the Desktop folder for the current user. If the setup is being run for All Users, and the ALLUSERS property is set, the DesktopFolder property should hold the full path to the All Users desktop folder.
FavoritesFolder	The FavoritesFolder property contains the full path to the Favorites folder for the current user.
FontsFolder	This property holds the full path to the Fonts folder.
IISROOTFOLDER	This property stores the root directory of the Web server on a target system. If you are using IIS functionality in your installation project and you have added a Web site, then IISROOTFOLDER is automatically added. For more information, see Adding IISROOTFOLDER Support.
INSTALLDIR	This property stores the default destination folder for your installation program's files. You can set an initial value for INSTALLDIR in the General Information view.
LocalAppDataFolder	The location of locally stored application data. A typical value of this property is: **C:\Users***UserName***\AppData\Local**
MyPicturesFolder	This property holds the full path to the current user's My Pictures folder.
PersonalFolder	This property holds the full path to the current user's Personal folder.
ProgramFiles64Folder	This property holds the full path to the 64-bit Program Files folder. This property requires Windows Installer version 2.0 or later.

Table 12-38 • Default Destination Folders for Basic MSI and InstallScript MSI Projects (cont.)

Folder Name	Description
ProgramFilesFolder	This property holds the full path to the 32-bit Program Files folder.
ProgramMenuFolder	This property is used to hold the full path to the Program menu for the current user. If the installation is being run for All Users, and the **ALLUSERS** property is set, the **ProgramMenuFolder** property should hold the full path to the All Users Programs menu.
SendToFolder	This property holds the full path to the current user's SendTo folder.
StartMenuFolder	This property is used to hold the full path the Start menu folder for the current user. If the installation is being run for All Users, and the **ALLUSERS** property is set, the **StartMenuFolder** property should hold the fully qualified path to the All Users program menu.
StartupFolder	This property is used to hold the full path to the Startup folder for the current user. If the setup is being run for All Users, and the **ALLUSERS** property is set, the StartupFolder property should hold the full path to the All Users program menu.
System16Folder	This property holds the full path to the folder containing the system's 16-bit .dll file. (In Windows Installer version 2.0, this property is no longer used.)
SystemFolder	This property holds the full path to the 32-bit Windows system folder.
System64Folder	This property holds the full path to the 64-bit Windows system folder. This property requires Windows Installer version 2.0 or later.
TempFolder	This property holds the full path to the Temp folder.
TemplateFolder	This property holds the full path to the current user's Templates folder.
WindowsFolder	This property holds the full path to the Windows folder.
WindowsVolume	This property holds the volume of the Windows folder. It is set to the drive where Windows is installed.

Folders in InstallScript Projects

The following folders are available only for InstallScript projects:

Table 12-39 • Default Destination Folders for InstallScript Projects

Folder Name	Description
Script-Defined Folders	This folder allows you to create folders whose location is determined in your script. Files cannot be added directly to this folder, but only to its subfolders; to create a subfolder, right-click the folder and select New Folder. Specify a folder name that is enclosed in angle brackets, for example, <MYFOLDER>; to assign the folder a location through your script, call FeatureSetTarget.
Application Target Folder	This folder's location is determined by the value of the system variable TARGETDIR.
Program Files	This folder's location is the 32-bit Program Files folder.
Program Files (64-bit)	This folder's location is the 64-bit Program Files folder.
Program Files\Common Files	This folder's location is the 32-bit Common Files folder.
Program Files (64-bit)\Common Files (64-bit)	This folder's location is the 64-bit Common Files folder.
Support Folder	This folder's location is determined by the value of the system variable SUPPORTDIR.
Windows	This folder's location is the Windows folder.
Windows\Fonts Folder	This folder's location is the Windows Fonts folder.
Windows\Windows System	This folder's location is the 32-bit Windows System folder.
Windows\Windows System (64-bit)	This folder's location is the 64-bit Windows System folder.

Redistributables View



Project • *The Redistributables view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Note that you can use the Prerequisites view and the Objects view to add redistributables to InstallScript projects.

The Redistributables view contains all of the InstallShield objects and third-party merge modules that are included with InstallShield. In Basic MSI and InstallScript MSI projects, this view also contains InstallShield prerequisites that you can add to your project.

InstallShield Prerequisites (Basic MSI and InstallScript MSI Projects)

An InstallShield prerequisite is an installation for a product or technology framework that is required by your product. You can add any of the existing InstallShield prerequisites to your installation projects and configure many of their settings. You can also create your own InstallShield prerequisites, and add them to your projects.

Including InstallShield prerequisites in your project enables you to chain multiple installations together, bypassing the Windows Installer limitation that permits only one Execute sequence to be run at a time. The **Setup.exe** setup launcher serves as a bootstrap application that manages the chaining.

InstallShield includes a base set of InstallShield prerequisites. You can also use the InstallShield Prerequisite Editor in InstallShield to define custom InstallShield prerequisites or to edit settings for any existing InstallShield prerequisites.

InstallShield includes support for two types of InstallShield prerequisites:

- **Setup prerequisite**—The installation for this type of prerequisite runs before your installation runs.
- **Feature prerequisite**—This type of prerequisite is associated with one or more features. It is installed if the feature that contains the prerequisite is installed and if the prerequisite is not already installed on the system. Thus, if a feature has a condition that is not met on the target system, or if the end user chooses not to install the feature, the feature is not installed. As a result, none of its associated feature prerequisites are installed, unless the feature prerequisites are also associated with other features that are installed.



Project *• Basic MSI projects include support for feature prerequisites.*

Merge Modules

A merge module (or .msm file) contains all of the logic and files needed to install distinct pieces of functionality. For example, many applications require Microsoft Visual Basic run-time .dll files. Instead of having to include the file in a component and figure out its installation requirements, you can simply attach the Visual Basic Virtual Machine merge module to one of your project's features.



Note *• Many of the merge modules included in the Redistributables view are authored by Microsoft or another third party. InstallShield distributes these modules as a courtesy to assist you in creating your installation project. However, InstallShield cannot modify or fix any problems that may exist within third party-authored modules. You are encouraged to contact the vendor regarding issues with specific third party-authored modules.*

InstallShield Objects

Like merge modules, objects contain logic and files needed to install distinct pieces of functionality. Some objects, such as the DirectX object included with InstallShield, require customization through a wizard. As soon as you add such an object to your installation, its customization wizard opens. You can either customize your object at the time you add it, or cancel the wizard and customize your object later by right-clicking the object and then clicking Change Objects Settings.

Redistributables Gallery

Because the file size of many of the redistributables is so large, some that are available for use in your projects are not added to your computer when you install InstallShield. However, these redistributables are still available for download from the Internet to your computer.

Configurable Merge Modules

A configurable redistributable is a merge module or an object that has at least one row in the **ModuleConfiguration** table that is referenced by at least one row in the **ModuleSubstitution** table. This enables you to change a value in the redistributable. When you select a configurable module in the Redistributables view, the Merge Module Configurable Values dialog box is displayed to enable you to configure the module at the time you add it. To customize the merge module later, right-click it and click select **Configure merge module**.

Working with the Redistributables View

The Redistributables view consists of the following elements:

- A row of buttons and other controls
- A group box area (below the row of buttons)
- A list of redistributables
- A details pane, which shows information about the selected redistributables

The following table describes all of the buttons and other controls that are displayed in the Redistributables view.

Table 12-40 • Controls in the Redistributables View

Name of Control	Icon	Description
Expand All Groups		Shows all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Collapse All Groups		Hides all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Show Details		Shows or hides the right pane in this view, which shows details about the InstallShield prerequisite, merge module, or object that is selected in the list of available redistributables. This details pane provides information such as which files a redistributable installs.
Show Group Box		Shows or hides the group box area below the row of buttons in this view.
Refresh		Refreshes the list of redistributables.
Search Grid		Dynamically filters the redistributables that are displayed in the Redistributables view according to the string that you specify in this search box. As you type a string in this box, InstallShield hides all of the rows that do not contain it.

Table 12-40 • Controls in the Redistributables View (cont.)

Name of Control	Icon	Description
Redistributables View Help		Displays the help for the Redistributables view.
Drag a column header here to group that column		Use this group box area to group rows in the view. The view supports multiple levels of grouping simply by dragging the column headings and dropping them onto the group box. InstallShield displays the rows in the view hierarchically according to column arrangement in the group box. To learn more, see Working with the Group Box Area in Various Views.

The following table describes each of the columns in the Redistributables view.

Table 12-41 • Columns in the Redistributables View

Column	Description
Check Box	This column shows a check box for each redistributable. To add a redistributable to your project, select its check box.
Type/Status	This column shows an icon that represents the type of redistributable, as well as its status. For details about each icon, see Managing the Redistributables Gallery.
Name	This column shows the name of each redistributable.
Version	This column shows the version number of the redistributable.
Type	This column lists the type of redistributable.
Location	This column indicates whether the redistributable is installed locally on your machine or it needs to be downloaded. For more information, see Downloading Redistributables to Your Computer.

Prerequisites View



Project • *The Prerequisites view is available in InstallScript projects.*

Note that you can use the Redistributables view to add InstallShield prerequisites to Basic MSI and InstallScript MSI projects.

The Prerequisites view contains all of the InstallShield prerequisites that are included with InstallShield. An InstallShield prerequisite is an installation for a product or technology framework that is required by your product. You can add any of the existing InstallShield prerequisites to your installation projects and configure many of their settings. You can also create your own InstallShield prerequisites, and add them to your projects.

InstallShield includes a base set of InstallShield prerequisites. You can also use the InstallShield Prerequisite Editor in InstallShield to define custom InstallShield prerequisites or to edit settings for any existing InstallShield prerequisites.

InstallShield includes support for two types of InstallShield prerequisites:

- **Setup prerequisite**—The installation for this type of prerequisite runs before your installation runs.
- **Feature prerequisite**—This type of prerequisite is associated with one or more features. It is installed if the feature that contains the prerequisite is installed and if the prerequisite is not already installed on the system. Thus, if a feature has a condition that is not met on the target system, or if the end user chooses not to install the feature, the feature is not installed. As a result, none of its associated feature prerequisites are installed, unless the feature prerequisites are also associated with other features that are installed.



Project • *Basic MSI projects include support for feature prerequisites.*

Redistributables Gallery

Because the file size of many of the redistributables is so large, some that are available for use in your projects are not added to your computer when you install InstallShield. However, these redistributables are still available for download from the Internet to your computer.

Working with the Prerequisites View

The Prerequisites view consists of the following elements:

- A row of buttons and other controls
- A group box area (below the row of buttons)
- A list of redistributables
- A details pane, which shows information about the selected redistributables

The following table describes all of the buttons and other controls that are displayed in the Prerequisites view.

Table 12-42 • Controls in the Prerequisites View

Name of Control	Icon	Description
Expand All Groups		Shows all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Collapse All Groups		Hides all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Show Details		Shows or hides the right pane in this view, which shows details about the InstallShield prerequisite that is selected in the list of available redistributables. This details pane provides information such as which files a redistributable installs.
Show Group Box		Shows or hides the group box area below the row of buttons in this view.

Table 12-42 • Controls in the Prerequisites View (cont.)

Name of Control	Icon	Description
Refresh		Refreshes the list of redistributables.
Search Grid		Dynamically filters the redistributables that are displayed in the Prerequisites view according to the string that you specify in this search box. As you type a string in this box, InstallShield hides all of the rows that do not contain it.
Prerequisites View Help		Displays the help for the Prerequisites view.
Drag a column header here to group that column		Use this group box area to group rows in the view. The view supports multiple levels of grouping simply by dragging the column headings and dropping them onto the group box. InstallShield displays the rows in the view hierarchically according to column arrangement in the group box. To learn more, see Working with the Group Box Area in Various Views.

The following table describes each of the columns in the Prerequisites view.

Table 12-43 • Columns in the Prerequisites View

Column	Description
Check Box	This column shows a check box for each redistributable. To add a redistributable to your project, select its check box.
Type/Status	This column shows an icon that represents the type of redistributable, as well as its status. For details about each icon, see Managing the Redistributables Gallery.
Name	This column shows the name of each redistributable.
Version	This column shows the version number of the redistributable.
Location	This column indicates whether the redistributable is installed locally on your machine or it needs to be downloaded. For more information, see Downloading Redistributables to Your Computer.

Objects View



Project • *The Objects view is available in the following project types:*

- *InstallScript*

- *InstallScript Object*

Note that you can use the Redistributables view to add merge modules to Basic MSI and InstallScript MSI projects.

The Objects view lets you add InstallScript objects and third-party merge modules to your projects.

Merge Modules

A merge module (or .msm file) contains all of the logic and files needed to install distinct pieces of functionality. For example, many applications require Microsoft Visual Basic run-time .dll files. Instead of having to include the file in a component and figure out its installation requirements, you can simply attach the Visual Basic Virtual Machine merge module to one of your project's features.



Note *• Many of the merge modules included in the Objects view are authored by Microsoft or another third party. InstallShield distributes these modules as a courtesy to assist you in creating your installation project. However, InstallShield cannot modify or fix any problems that may exist within third party-authored modules. You are encouraged to contact the vendor regarding issues with specific third party-authored modules.*

InstallScript Objects

InstallShield enables you to create your own InstallScript objects that can be used in any of your installation projects or distributed for use by other installation developers.

Redistributables Gallery

Because the file size of many of the redistributables is so large, some that are available for use in your projects are not added to your computer when you install InstallShield. However, these redistributables are still available for download from the Internet to your computer.

System Configuration View



Project *• The System Configuration view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Every installation changes the target system in some way. The simplest installations might only copy files. More in-depth installations make registry changes, edit .ini files, or create shortcuts. InstallShield provides the following views for making advanced configurations.

Shortcuts



Project *• The Shortcuts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Shortcuts offer quick access to your installed application. You can create shortcuts and program folders on the desktop, the Start menu, and various other locations.

Registry



Project *• The Registry view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*
- *QuickPatch*

In the Registry view, you can create registry entries or import existing registry data into your setup.

ODBC Resources



Project *• The ODBC Resources view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

In the ODBC Resources view, you can add drivers, translators, and data sources to one or more of your application's features. Select from installed ODBC resources or add new ones to the list, associate them with features, and then customize their attributes.

INI File Changes



Project *• The INI File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The InstallScript language includes several initialization file functions for manipulating .ini files.

Although editing system .ini files is not recommended, you can edit these files in the INI File Changes view. This view enables you to create sections and keywords, or edit an .ini file on the target system.

Environment Variables



Project *• The Environment Variables view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The InstallScript language includes the GetEnvVar function for retrieving the current value of an environment variable, and it enables you to create an environment variable.

If you need to create, modify, or remove environment variables from the target system, you can define their settings in the Environment Variables view.

XML File Changes



Project *• The XML File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

Sometimes you need to modify XML files that store settings related to your product as well as standard configuration files like **web.config** and **machine.config**. You can use the XML File Changes view in InstallShield to specify any XML file changes that should be made on the target system.

Text File Changes



Project *• The Text File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The InstallScript language includes string functions for finding and modifying string variables and literals.

The Text File Changes view is where you configure search-and-replace behavior for content in text files—for example, .txt, .htm, .xml, .config, .ini, and .sql files—that you want to modify at run time on the target system. The text files can be part of your installation, or they can be files that are already present on target systems.



Tip *• The Text File Changes view enables you to modify XML files without using MSXML (which is a run-time requirement if you use the XML File Changes view to modify XML files).*

Scheduled Tasks



Project *• The Scheduled Tasks view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The Scheduled Tasks view is where you create and configure automated tasks that you want to be created through the Windows task scheduler at run time on target systems. The files that you want to be launched by the scheduled tasks can be part of your installation, or they can be files that are already present on target systems.

Services



Project • *The Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The Services view is where you specify whether to install, configure, start, stop, and uninstall Windows services during installation and uninstallation.

As an alternative, you can use the Services area under the Advanced Settings node of the Components view and the Setup Design view to install, configure, start, stop, and uninstall Windows services. If you configure service information in either of these areas, the other area is automatically updated accordingly.

Shortcuts View



Project • *The Shortcuts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Shortcuts view provides a simple, visual way to create shortcuts to your application on the target system. This view contains a Shortcuts explorer that shows a set of predefined destination folders under which you can create shortcuts and subfolders. InstallShield offers the following standard shortcut destinations.

Table 12-44 • Standard Shortcut Destinations

Shortcut Destination	Project Type	Description
Programs Menu and Startup	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	The Programs Menu and Startup folders are located on the Start menu. The Programs Menu folder is the industry-standard place for installing product shortcuts. The Startup folder should contain shortcuts to only those items that need to be launched whenever Windows starts.

Table 12-44 • Standard Shortcut Destinations (cont.)

Shortcut Destination	Project Type	Description
Send To	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	The Send To folder is available when an end user right-clicks a file; the context menu that is displayed contains a command called Send To. If you create a shortcut for your program in this folder, an end user can click any file and send it to your product. For example, you might want your end users to be able to open an HTML page in Notepad. If you created a shortcut to Notepad in the Send To folder in the Shortcuts view, end users could right-click an HTML file and click Notepad from your Send To menu. The source file for that page opens in Notepad.
Desktop	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	The Desktop folder contains shortcuts for the end user's desktop. When you create a shortcut in the Desktop folder, your product's icon is displayed on the end user's desktop. ***Note*** • *The desktop is the most visible place to put a shortcut, but too many shortcut icons can clutter the end user's desktop.*

When you add a new shortcut, InstallShield adds it as a new node to the Shortcuts explorer in this view. The new shortcut node is displayed with the same icon that is selected in the Icon File setting for the shortcut—which is the icon that is used when the shortcut is added to target systems at run time—unless either of the following conditions is true:

- The project type is InstallScript.
- The shortcut is for a preexisting file on the target system.

For both of the aforementioned conditions, the icon file is not known until run time. Therefore, InstallShield shows the following icon for each shortcut in the Shortcuts explorer, instead of displaying the icon that is used at run time on target systems.



InstallShield also uses this icon for a shortcut in the Shortcuts explorer if the icon file does not contain an icon.

Note • *You cannot create shortcuts to a dynamically linked file. For more information, see Limitations of Dynamic File Linking.*

For details about each of the settings that you can configure for shortcuts, folders, as well as tiles, see:

- Shortcut Settings
- Folder Settings
- Tile Configuration Settings

Shortcut Settings



Project *• The Shortcuts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The settings for a shortcut are organized into the following main categories in the Shortcuts view:

- Appearance
- Behavior
- General
- UWP App Package Tile Overrides



Edition *• UWP App Package Tile Overrides settings are available in Basic MSI projects edited with the Premier edition only. InstallScript MSI, direct-mode projects (Windows Installer Databases, Transforms), and Professional or Express editions will not see the UWP app-related options in the* ***Releases*** *view or the* ***Shortcuts*** *view.*

Appearance

Use a shortcut's Appearance settings to specify details such as the display name and the icon for the shortcut.

Table 12-45 • Appearance Settings

Setting	Project Type	Description
Display Name	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter the name of the shortcut as it should appear on the target system. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** *• For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—You must enter a value for this setting; otherwise, a build error occurs.* *The display name is of the Windows Installer Filename data type. If the display name that you enter is not already in the 8.3 format, InstallShield uses the ShortName|LongName format for this setting. For example, if you enter* ***My Product Name*** *as the value in this setting, InstallShield sets the value to* ***MyProd~1|My Product Name****, or something similar, so that a short name is available at run time if needed.*

Table 12-45 • Appearance Settings (cont.)

Setting	Project Type	Description
Display Resource	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	If you are preparing an installation for a multilingual application and you are separating language-neutral portable executable files from .mui files for the display name of your shortcut, use the following settings: • **Display Resource**—Click the ellipsis button (...) in this setting if you want to browse to the DLL file that contains the multilingual user interface (MUI) manifest. InstallShield lists the path and file name in this setting if you do either of the following: you select a DLL file by browsing for it, or you manually enter a path and file name in the Display Resource DLL setting. InstallShield also lists the resource index that is specified in the Display Resource Index setting. If this setting contains a value, the value for the Display Name setting is ignored. If you leave this setting blank, Windows Installer uses the value for the Display Name setting. • **Display Resource DLL**—If you want to manually specify the path and file name of the DLL file that contains the MUI manifest, enter it. You can include Windows Installer directory properties—for example, **[INSTALLDIR]MyResource.dll**—instead of hard-coded directory paths. If you click the ellipsis button in the Display Resource setting to browse to the DLL file, InstallShield uses the format [#filekey] in the Display Resource DLL setting to identify the DLL file. • **Display Resource Index**—Specify the display name index for the shortcut. This must be a non-negative number.  ***Note*** • *If you specify a DLL, you must also enter a value for the Display Resource Index setting.* These settings enable you to separate language-neutral portable executable files from .mui files, which contain all of the language-dependent resources, and later add resources for additional languages without having to recompile or relink the application. Windows Vista and later and Windows Server 2008 and later include support for the display resource. Earlier systems ignore it.

Table 12-45 • Appearance Settings (cont.)

Setting	Project Type	Description
Description	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter a description of the shortcut. The text that you enter is displayed as a tooltip when the end user places the mouse pointer over the shortcut. It is also displayed in the Description field of the shortcut's Properties dialog box. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 12-45 • Appearance Settings (cont.)

Setting	Project Type	Description
Description Resource	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	If you are preparing an installation for a multilingual application and you are separating language-neutral portable executable files from .mui files for the description of your shortcut, use the following settings: • **Description Resource**—Click the ellipsis button (...) in this setting if you want to browse to the DLL file that contains the multilingual user interface (MUI) manifest. InstallShield lists the path and file name in this setting if you do either of the following: you select a DLL file by browsing for it, or you manually enter a path and file name in the Description Resource DLL setting. InstallShield also lists the resource index that is specified in the Description Resource Index setting. If this setting contains a value, the value for the Description setting is ignored. If you leave this setting blank, Windows Installer uses the value for the Description setting. • **Description Resource DLL**—If you want to manually specify the path and file name of the DLL file that contains the MUI manifest, enter it. You can include Windows Installer directory properties—for example, **[INSTALLDIR]MyResource.dll**—instead of hard-coded directory paths. If you click the ellipsis button in the Description Resource setting to browse to the DLL file, InstallShield uses the format [#filekey] in the Description Resource DLL setting to identify the DLL file. • **Description Resource Index**—Specify the description index for the shortcut. This must be a non-negative number. ***Note*** • *If you specify a DLL, you must also enter a value for the Description Resource Index setting.* These settings enable you to separate language-neutral portable executable files from .mui files, which contain all of the language-dependent resources, and later add resources for additional languages without having to recompile or relink the application. Windows Vista and later and Windows Server 2008 and later include support for the description resource. Earlier systems ignore it.

Table 12-45 • Appearance Settings (cont.)

Setting	Project Type	Description
Icon	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	To specify the icon that you want to be displayed for the shortcut, use the following settings: • **Icon**—Specify the file that contains the icon for the shortcut that you are creating. You must specify an .ico file or the executable file (.dll or .exe) that contains the icon resource. InstallShield lists the icon path and index in the Icon setting if you do either of the following: you select a file by browsing for it, or you manually enter a path and file name in the Icon File setting. The method that you use to specify the file depends on the type of project that you are using.  ***Project*** *• For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—Click the ellipsis button (...) in this setting if you want to browse to the .ico file or the .exe or .dll file that contains the icon resource.* *For InstallScript projects—Click the ellipsis button (...) in this setting if you want to browse to the location on the target system of the .ico file or the .exe or .dll file that contains the icon resource.* • **Icon File**—If you want to manually specify the path and name of the file that contains the icon, enter it. • **Icon Index**—If the icon file that you specify contains more than one icon resource, enter the index in this setting. A nonnegative integer refers to the order of the icon resources in the executable file. For example, 0 refers to the first icon in the file, 1 refers to the second icon, and 2 refers to the third icon. Use a negative number to refer to a specific resource ID. For example, the icon index -12 points to the icon with a resource ID of 12.  ***Project*** *• For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—Note that because Windows Installer requires a separate icon file when the component is to be advertised, InstallShield extracts the first icon from the .exe or .dll file unless an icon index is specified. If you do not specify an icon file, the key file of this component is automatically used for the shortcut's icon.*

Behavior

Use a shortcut's Behavior settings to specify details such as the target and keyboard shortcut.

Table 12-46 • Behavior Settings

Setting	Project Type	Description
Advertised	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify whether you want to create an advertised shortcut. If you select Yes and the end user advertises the product or the feature that contains the shortcut, the shortcut is created but the component's files are not installed until the end user launches the shortcut. The first time that the shortcut is launched, the Windows Installer installs the component's files and other data, and then the shortcut launches the target file. Every time that the shortcut is used from then on, it behaves like a normal shortcut.  ***Note*** • *To create an advertised shortcut, the shortcut's component must have a key file. The key file is the shortcut's target.*
Target	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter the path to the file on the target system that should be launched when end users launch the shortcut. As an alternative to manually entering a value, you can click the ellipsis button (...) to browse to the shortcut target.  ***Project*** • *For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—Use Windows Installer directory properties—for example,* ***[INSTALLDIR]File.exe****—instead of hard-coded directory paths. If the shortcut is an advertised shortcut, this setting is read-only. The key file of the shortcut's component is automatically set as the shortcut's target. If the shortcut's component does not have a key file, the shortcut will not function properly on the target system.* *For InstallScript projects—Use system variables—for example,* ***<TARGETDIR>\File.exe****—instead of hard-coded directory paths.*

Table 12-46 • Behavior Settings (cont.)

Setting	Project Type	Description
Arguments	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter the command-line arguments for the shortcut. The arguments are added to the Target value on the shortcut's Properties dialog box on the target system. These arguments work in the same way as any other command-line arguments. For example, you can link a file to an executable file or cause an executable file to run silently by passing command-line arguments.  ***Note*** • *Verify that the syntax is correct because InstallShield does not do this.*  ***Tip*** • *Use* **%1** *in the argument for the selected file name. For example, if the end user right-clicks the file* **C:\File.ext** *and* `-p %1` *is the argument for this shortcut, the command-line argument becomes* `-p C:\File.ext`. *In some cases, it is necessary to enclose the %1 argument in quotation marks—as in* **"%1"***—to correctly handle file names that contain spaces.*
Working Directory	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter the working directory for the shortcut target, or click the ellipsis button (...) to select or create a directory. The directory that you specify is displayed as the **Start in** field on the shortcut's Properties dialog box on the target system. The working directory is the default directory that is displayed in standard file-opening and file-saving dialog boxes, as well as the current directory used by the product.  ***Project*** • *For example, in Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects, if you want your working directory to be set to a subdirectory of* ***INSTALLDIR*** *called* ***Files****, select* ***[INSTALLDIR]*** *from the list and add the subdirectory name* **Files** *to the end of it. When you are finished, it should read* ***[INSTALLDIR]Files****.* *In an InstallScript project, if you want your working directory to be set to a subdirectory of TARGETDIR called* ***Files****, select* ***<TARGETDIR>*** *from the list and add the text* **\Files** *to the end of it. When you are finished, it should read* ***<TARGETDIR>\Files****.*

Table 12-46 • Behavior Settings (cont.)

Setting	Project Type	Description
Hot Key	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting contains the decimal value of the keyboard shortcut that is assigned to your shortcut. You can assign a keyboard shortcut to your product's shortcut so that end users can press the appropriate hot keys to launch the shortcut. If you want InstallShield to calculate the decimal value of the keyboard shortcut, click the ellipsis button (...) in this setting. For more information, see Specifying a Keyboard Shortcut for Accessing a Shortcut.  ***Caution*** • *It is recommended that you avoid configuring keyboard shortcuts for your shortcuts because they may conflict with existing keyboard shortcuts on the target system.*
Run	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Select the style of window that the target file should use when end users launch the shortcut. Available options are: • **Normal Window**—The file is launched in a standard-sized window. • **Maximized Window**—The file is launched in full-screen view. • **Minimized Window**—The file is launched in a minimized window, visible only on the taskbar.
Prevent Pinning	InstallScript	Specify whether you want to prevent end users from being able to pin a shortcut to the taskbar and to the Start menu after end users install your product. Available options are: • **Yes**—When the shortcut is installed, the context menu commands for pinning a shortcut to the taskbar and to the Start menu are hidden. • **No**—When the shortcut is installed, the context menu commands for pinning the shortcut to the taskbar and to the Start menu are available, enabling end users to pin the shortcut. This is the default option. Windows 7 introduced support for this setting. Earlier versions of Windows ignore this setting. You may want to disable pinning for shortcuts that are for tools and secondary products that are part of your installation. Note that if you configure the shortcut to prevent this pinning, the target of the shortcut is ineligible for inclusion in the most frequently used list on the Start menu. For more information, see Specifying Whether End Users Should Be Able to Pin a Shortcut to the Taskbar or Start Menu.

Table 12-46 • Behavior Settings (cont.)

Setting	Project Type	Description
Do Not Highlight as New	InstallScript	Specify whether you want to prevent a shortcut on the Start menu from being highlighted as newly installed after end users install your product. Available options are: • **Yes**—If the installation adds the shortcut to the Start menu, the shortcut is not highlighted as newly installed. This has the same effect as clearing the **Highlight newly installed programs** check box in the Customize Start Menu dialog box for an individual item on a target system. • **No**—If the installation adds the shortcut to the Start menu, the shortcut is highlighted as newly installed. This is the default option. Windows 7 introduced support for this setting. Earlier versions of Windows ignore this setting. You may want to select Yes for shortcuts that are for tools and secondary products that are part of your installation. For more information, see Preventing a Shortcut on the Start Menu from Being Highlighted as Newly Installed.
Allow Initial Pin to Start	InstallScript	Specify whether you want the shortcut to be pinned by default to the Start screen on Windows 8 target systems. Available options are: • **Yes**—When the shortcut is installed on Windows 8 systems, it is pinned to the Start screen. End users can optionally unpin the shortcut. This is the default option. • **No**—When the shortcut is installed on Windows 8 systems, it is not pinned to the Start screen. It is available in the Apps list that contains shortcuts to all of the applications on the system. For more information, see Suppressing Initial Pinning of a Shortcut to the Windows 8 Start Screen.

General

Use a shortcut's General settings to specify details such as the component that contains the shortcut.

Table 12-47 • General Settings

Setting	Project Type	Description
Key Name	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter a key name for the shortcut. The key name is an internal name; it is not displayed to end users.

Table 12-47 • General Settings (cont.)

Setting	Project Type	Description
Internal Name	InstallScript	Enter an internal name for the shortcut. The internal name is not displayed to end users.
Component	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting indicates the component that contains the shortcut. Click the ellipsis button (...) to display the Browse for a Component dialog box, which lets you create new components or select an existing component for the shortcut. The shortcut is created on the target system if the selected component is installed.  ***Project*** • *In InstallScript projects, you can select more than one component for the shortcut.*
Feature	Basic MSI, InstallScript MSI, MSI Database, Transform	This setting indicates the feature or features with which the shortcut component is associated. If there is no feature-component association, this setting is blank.
Uninstall	InstallScript	Specify whether the shortcut should be removed when the product is uninstalled.
Replace Existing (If Found)	InstallScript	Specify whether the shortcut should overwrite an already existing shortcut with the same display name in the same location on the target system.
Internet Shortcut	InstallScript	Specify whether the shortcut is an Internet shortcut. The default setting is No; if you select Yes, the Target setting for the shortcut should be a valid URL.
VS .NET Project Output	InstallScript	Specify whether the shortcut points to a Visual Studio project output.
Type	InstallScript	Specify how the shortcut should be created. Available options are: • **Automatic**—The installation automatically determines where the shortcut should appear based on the value of the *ALLUSERS* system variable. If *ALLUSERS* is non-zero, the shortcut is created in the All Users profile so that it is available on the target system for all end users. If *ALLUSERS* is FALSE, the shortcut is created in the current user's profile. • **Personal**—The shortcut is created in the current user's profile. • **Common**—The shortcut is created in the All Users profile.

Table 12-47 • General Settings (cont.)

Setting	Project Type	Description
Comments	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter comments for this shortcut. Comments are saved in the project file for your reference and are not displayed to the end user.

Table 12-47 • General Settings (cont.)

Setting	Project Type	Description
Shell Properties	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting lets you specify one or more shortcut properties that need to be set by the Windows Shell at run time. To specify one or more properties, click the Add New Shell Shortcut Property button in this setting, and then click one of the available options: • **Prevent Pinning**—The context menu commands for pinning the shortcut to the taskbar or the Start menu are not displayed for this shortcut after end users install your product. In addition, this option makes the target of the shortcut ineligible for inclusion in the most frequently used list on the Start menu. If you select this option, InstallShield adds a Key Name setting, plus additional rows of subsettings, and configures them as needed. • **Do Not Highlight as New**—The Start menu entry for the shortcut is not highlighted as newly installed after end users install your product. This has the same effect as clearing the **Highlight newly installed programs** check box in the Customize Start Menu dialog box for an individual item on a target system. You may want to set this property for shortcuts that are for tools and secondary products that are part of your installation. If you select this option, InstallShield adds a Key Name setting, plus additional rows of subsettings, and configures them as needed. • **Suppress Initial Pin to Windows 8 Start Screen**—If you select this option, InstallShield suppresses the Windows 8 default behavior of pinning a shortcut to the Start screen during initial installation on Windows 8 target systems. The installation sets a Windows Shell property that was introduced in Windows 8. You may want to suppress pinning for shortcuts that are tools and secondary products that are part of your installation. • **Custom Property**—To specify a custom property and value for shortcut properties that need to be set by the Windows Shell at installation run time, select this option. InstallShield adds a new Key Name setting, plus additional subsettings, that let you configure the appropriate property name and value. Specify additional properties as needed.  ***Note*** • *Windows Installer 5 introduced support for the Shell property settings. Earlier versions of Windows Installer ignore these settings.* For more information, see Setting Shell Properties for a Shortcut.

Table 12-47 • General Settings (cont.)

Setting	Project Type	Description
Key Name	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting is displayed if you have used the Shell Properties setting to add one or more Shell properties to your shortcut. Enter a key name for the Shell shortcut property. The key name is an internal name; it is not displayed to end users. ***Note*** • *Windows Installer 5 introduced support for the Shell property settings. Earlier versions of Windows Installer ignore these settings.* For more information, see Setting Shell Properties for a Shortcut.
Property	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting is displayed if you have used the Shell Properties setting to add one or more Shell properties to your shortcut. Enter the shortcut property that you want the Windows Shell to set at run time. ***Note*** • *Windows Installer 5 introduced support for the Shell property settings. Earlier versions of Windows Installer ignore these settings.* For more information, see Setting Shell Properties for a Shortcut.
Value	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	This setting is displayed if you have used the Shell Properties setting to add one or more Shell properties to your shortcut. Enter the value for the Shell shortcut property. ***Note*** • *Windows Installer 5 introduced support for the Shell property settings. Earlier versions of Windows Installer ignore these settings.* For more information, see Setting Shell Properties for a Shortcut.

UWP App Package Tile Overrides

Edition • *UWP App Package Tile Overrides settings are available in Basic MSI projects edited with the Premier edition.*

Use a shortcut's UWP App Package Tile Overrides settings to configure tiles created in the UWP app package. Refer to the following table for descriptions of these settings. Refer to the UWP App Logo Considerations subsection for additional information.

Table 12-48 • UWP App Package Tile Overrides Settings

Setting	Project Type	Description
Include in UWP App Package	Basic MSI	Specify whether you want this shortcut to be included in all UWP app packages created from this project. • **Yes**—Include this shortcut in all UWP app packages created from this project. The shortcut will be turned into a tile if it points to an .exe installed by the UWP app package. • **No**—This shortcut will be excluded from all UWP app packages created from this project.
Short Name	Basic MSI	Specify a short string to appear on a medium or larger size tile created in the UWP app package. If nothing is specified, the name is copied from the **Display Name** setting of the shortcut.
Background Color	Basic MSI	Select a color to be used as the background color of the tile created in the UWP app package. A drop-down selection provides several predefined colors, or an ellipsis button (...) lets you choose additional colors or specify an RGB hex color. If nothing is specified, the transparency that is set by Windows is used as the default tile background.
Small Logo	Basic MSI	Specify a 44x44 image file to use as a small logo icon that is displayed in the left column of the Start Menu's All Apps list. If an image file is not provided, InstallShield generates a small logo from the target's tile configuration or the shortcut's icon.

Medium Logo	Basic MSI	Specify a 150x150 image file to use as a medium tile when pinned and to use as the scaled-down small tile. A medium-sized logo image is required; if an image file is not provided, InstallShield generates a medium logo from the target's tile configuration or the shortcut's icon.
		The following subsetting is available:
		• **Show Name on Medium Logo**—Specify whether you want to show a name of the application on a medium tile created in the UWP app package. Available options are:
		  ◦ **Yes**—Display the name specified in the **Short Name** field over the logo image on a medium tile. If nothing is specified for **Short Name** the name is pulled from the **Display Name** setting of the shortcut.
		  ◦ **No**—No name will be displayed on the medium tile created in the UWP app package.

Table 12-48 • UWP App Package Tile Overrides Settings (cont.)

Setting	Project Type	Description

Large Logo	Basic MSI	Specify a 310x310 image file to be used for a large tile, only when pinned and resized. This setting is optional. If specified, this will allow the tile to be set to Large.
		
		The following subsetting is available:
		
		• **Show Name on Large Logo**—Specify whether you want to show a name of the application on a large tile created in the UWP app package. Available options are:
		
		  ◦ **Yes**—Display the name specified in the **Short Name** field over the logo image on a large tile. If nothing is specified for **Short Name** the name is pulled from the **Display Name** setting of the shortcut.
		  ◦ **No**—No name is displayed on the large tile created in the UWP app package.

Wide Logo	Basic MSI	Specify a 310x150 image file to be used for a wide tile, only when pinned and resized. This setting is optional. If specified, this will allow the tile to be set to Wide.
		
		The following subsetting is available:
		
		• **Show Name on Large Logo**—Specify whether you want to show a name of the application on a large tile created in the UWP app package. Available options are:
		
		  ◦ **Yes**—Display the name specified in the **Short Name** field over the logo image on a wide tile. If nothing is specified for **Short Name** the name is pulled from the **Display Name** setting of the shortcut.
		  ◦ **No**—No name is displayed on the wide tile created in the UWP app package.

UWP App Logo Considerations

Important • *InstallShield must be installed on a Windows 10 machine or a machine with the Windows 10 SDK installed on it in order to utilize UWP app-related image variants described in this section. If the Windows 10 SDK and in particular* ***MakePRI.exe*** *cannot be found on the build machine, image variants are collected anyway, but no* ***resources.pri*** *file can be generated. Without the* ***resources.pri*** *file created by* ***MakePRI****, Windows only uses the neutral image variant. The neutral image variant has no qualifiers in its name, and is created as a scale-100 size image.*

The following table provides UWP app logo considerations.

Topic	Consideration

Logo Sizes	All logos are expected to be of certain sizes. In addition, multiple contrast variants and scale factors (DPI-based scaling) can be provided in a mix of two ways. For each variant that is found, InstallShield will include the file, converting or resizing it as necessary. The scanning process is similar to that used by InstallShield's suite High DPI support. Both implementations are inspired by the process Windows uses at run time, described in Globalization and accessibility for tile and toast notifications (Windows Runtime apps):
	• Related image files next to a source image will be scanned. For example, if including a file called **<images>\Logo.png**, InstallShield will look for that file as the unqualified file, and several other qualified variants:
	  ◦ **<images>\contrast-black\scale-nnn\Logo.png**
	  ◦ **<images>\contrast-white\Logo.scale-nnn.png**
	  ◦ **<images>\scale-nnn\Logo.png**
	  ◦ **<images>\Logo.scale-nnn.png**
	• If the original source is an icon resource of an .exe or .dll, and a more appropriate scale variant is not found per the above naming conventions, InstallShield will use the original as the source for any scale.

Scale Factors	Scale factors result in different required sizes. For example, a Large Logo is normally 310x310, but at scale-150, it is 465x465 (310 * 150 / 100 = 465). InstallShield currently scans for and includes scale-100, scale-125, scale-150, scale-200, and scale-400 images. You do not have to have an actual image without the scale-*nnn* component, but it makes using InstallShield's file browse a lot simpler.
Source Image Type	If a specified source image is not a .png or .jpg file, or is the wrong size, it will be converted to a .png of the correct size at build time.  ***Note*** *• Converting an icon resource at multiple scales may result in different size icon sources being in use for each different scale. For example, a scale-200 Medium Logo is 300x300, which would use a 256x256 icon that the scale-100 Medium Logo likely would not.*
Logo Image Layout Recommendations	There are recommendations on logo image layouts from Microsoft with additional detail in Guidelines for tile and icon assets. Microsoft suggests a certain amount of padding around the core portion of a logo. For example, on the Medium tile (150x150), they suggest 45 pixels or 33% padding, resulting in a 60x60 (or 50x50) core area for the logo. InstallShield does nothing to follow or enforce this recommendation.

Folder Settings



Project *• The Shortcuts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The following settings are available for a folder in the Shortcuts view.

Table 12-49 • Folder Settings

Setting	Project Type	Description
Key Name	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter a key name for the folder. The key name is the internal name that is used for the folder in the Directory column of the **Directory** table; it is not displayed to end users.
Internal Name	InstallScript	Enter an internal name for the folder. The internal name is not displayed to end users.

Table 12-49 • Folder Settings (cont.)

Setting	Project Type	Description
Display Name	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter the name of the folder as it should appear on the target system. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Project*** *• For Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects—You must enter a value for this setting; otherwise, a build error occurs.* *The display name is of the Windows Installer DefaultDir data type. If the display name that you enter has more than eight characters, InstallShield uses the ShortName|LongName format for this setting. For example, if you enter* ***My Product Name*** *as the value in this setting, InstallShield sets the value to* ***MyProd~1|My Product Name****, or something similar, so that a short name is available at run time if needed.* *For InstallScript and InstallScript MSI projects—You can enter the group name system variable* **<SHELL_OBJECT_FOLDER>** *as the display name for your folder. You can then define the display name of this folder at run time by setting the system variable SHELL_OBJECT_FOLDER in InstallScript. To use this functionality in an InstallScript MSI installation, any letters that are specified for the Key Name setting of the folder must be all uppercase (for example,* ***NEWFOLDER1****). For more information, see SHELL_OBJECT_FOLDER.*
Description	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Enter a description of the folder.
Shared	InstallScript	Specify whether this file is shared with other installations: • **Yes**—If this folder contains any subfolders or files that were not created by the installation, the folder is not removed from the target system when your product is uninstalled. • **No**—When your product is uninstalled from the target system, this folder is removed.

Tile Configuration Settings



Project • *Tile Configuration Settings are available within the Shortcuts view for the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

The settings for tile configurations are organized into the following main categories:

- General
- Appearance

General

Use the tile configuration's General setting to view the path of the .exe file that is associated to the tile.

Table 12-50 • General Settings

Setting	Project Type	Description
Target	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Displays the path of the .exe file (for example, **[INSTALLDIR]File.exe**) that is associated to the tile.

Appearance

Use a tile configuration's Appearance settings to specify tile customizations such as the background color of the tile.

Table 12-51 • Appearance Settings

Setting	Project Type	Description
Text Color	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Choose a foreground text color that is used to display the app name on a medium-sized tile. Available drop-down options are: • **Light**—Sets the text color that is displayed on a medium-sized tile to be light (white text). • **Dark**—Sets the text color that is displayed on the medium-sized tile to be dark (black text).  ***Note*** • *Only a medium-sized tile (150x150) has room to display a name on the tile. Also, in order for the* ***Text Color*** *setting to have an effect, the* ***Show Name on Medium Tile*** *setting must be* ***Yes****.*
Background Color	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify a color to be used as the background color of the tile. This color must be in RGB hex string format #rrggbb, or must be a known color. A drop-down selection provides a list of known colors, or an ellipsis button (...) lets you choose a custom color interactively.

Table 12-51 • Appearance Settings (cont.)

Setting	Project Type	Description
Small Tile Image	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify a relative path to a small-sized (70x70) tile background image, or click the ellipsis button (...) to browse to a .png or .jpg file.  ***Important*** *• If you choose to use custom images, refer to the following considerations:* • *Custom images must be added to the project manually and you must specify an image for both the* ***Small Tile Image*** *(70x70) and the* ***Medium Tile Image*** *(150x150). If you specify only one, the default Windows styling for the app icon and background is applied to the tile.* • *If you provide scaled, high-contrast, or localized images, you need to manually create a* ***resources.pri*** *file and place it in the same directory as the app's* ***.exe*** *file and its* ***.visualelementsmanifest.xml*** *file. For more information about using images or creating a* ***resources.pri*** *file, refer to the following MSDN article: How to customize Start screen tiles for desktop apps (Windows Runtime apps).* • *When browsing for a tile image, if you have created or will create a* ***resources.pri*** *file, the specific image selected may include qualifiers that the* ***resources.pri*** *file maps to (such as ISLogo.****scale-80****.png). In this case, the qualifiers should be removed from the value for* ***Small Tile Image*** *or* ***Medium Tile Image****. However if a* ***resources.pri*** *is not being used or doesn't have a mapping for this particular file, then the qualifiers are part of the file name and should be retained. If qualifiers are detected, a dialog appears to let you choose which path and file name to use.*

Table 12-51 • Appearance Settings (cont.)

Setting	Project Type	Description
Medium Tile Image	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify a relative path to a medium-sized (150x150) tile background image. ***Important*** *• If you choose to use custom images, refer to the following considerations:* • *Custom images must be added to the project manually and you must specify an image for both the* ***Small Tile Image*** *(70x70) and the* ***Medium Tile Image*** *(150x150). If you specify only one, the default Windows styling for the app icon and background is applied to the tile.* • *If you provide scaled, high-contrast, or localized images, you need to manually create a* ***resources.pri*** *file and place it in the same directory as the app's* ***.exe*** *file and its* ***.visualelementsmanifest.xml*** *file. For more information about using images or creating a* ***resources.pri*** *file, refer to the following MSDN article: How to customize Start screen tiles for desktop apps (Windows Runtime apps).* • *When browsing for a tile image, if you have created or will create a* ***resources.pri*** *file, the specific image selected may include qualifiers that the* ***resources.pri*** *file maps to (such as ISLogo.****scale-80****.png). In this case, the qualifiers should be removed from the value for* ***Small Tile Image*** *or* ***Medium Tile Image****. However if a* ***resources.pri*** *is not being used or doesn't have a mapping for this particular file, then the qualifiers are part of the file name and should be retained. If qualifiers are detected, a dialog appears to let you choose which path and file name to use.*
Show Name on Medium Tile	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Specify whether the app name should be shown on a medium-sized tile (150x150). Available options are: • **Yes**—Display the app name on the tile when a medium-sized tile is used. • **No**—Do not display the app name on the tile. ***Note*** *• This setting has no effect when a small-sized tile is used.*

Registry View

Project *• The Registry view is available in the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Registry view enables you to define registry keys and values to be created by your installation. This view has four different panes. The two top panes show the registry data contained on your development system, and the two bottom panes are where you set up the registry data to be created on the target system.

For Windows Installer projects (Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform), you can use the View Filter list at the top of the view to select the component whose registry data you want to display.

All registry data (except the <Default> registry set in a InstallScript project) must be associated with a component. This way, for installation projects, if the component's feature is selected for installation, the component's registry data are set up on the target system.

In InstallScript projects, groups of registry keys and values are called registry sets. There is no limit to the number of registry sets you can create (or the number of keys and values inside a single registry set).

The installer automatically creates certain registry entries based on values you provide in the General Information view. These informational keys are required by Windows logo guidelines. Also, all of a component's advanced settings are used to register files on the target system.

ODBC Resources View



Project • *The ODBC Resources view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

One of the more complex areas of system configuration involves setting up ODBC drivers, data source names (DSNs), and translators. The ODBC resource must be properly registered on the system with all of the required attributes and, in the case of drivers and translators, install the necessary files, including any installation .dll files. This process is simplified in the ODBC Resources view, in which you can select the drivers, data sources, and translators installed on your development system.

For details about each of the settings that you can configure for ODBC resources, see ODBC Resource Settings.

Project *• For installation projects: The ODBC Resources view is exclusively for installing ODBC-related resources. To install the core ODBC files, select the MDAC 2.5 merge module in the Redistributables view.*

For DIM and Merge Module projects: The Merged Feature check box in the Associated Feature(s) pane of this view is selected when you add an item and cleared when you remove an item in the ODBC Resources pane. It does not affect your project.

ODBC Resource Settings

Project *• The ODBC Resources view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

There is no universal list of ODBC resource attributes and their possible values. Check with the file's vendor or author if you are unsure about what to specify for its installation. Some attributes are common to each type of ODBC resource, as described below.

Note *• You can add your own ODBC attribute by clicking in the last row of the grid in the ODBC Resources view and specifying the property and value.*

Drivers

In addition to the required attributes below, the driver must have a name in the tree. This name is registered as the driver's description. The driver's name cannot be localized, which means that it does not appear as one of your project's string entries. Because of this, the same value is used regardless of the system's language.

Table 12-52 • ODBC Driver Attributes

Attribute	Expected Value
Driver Component	Enter the name of the component that you want to install the selected ODBC driver and its attributes.
Driver DLL	Enter the path to the DLL file that is on the development system and that serves as the ODBC driver, or click the ellipsis button (...) to browse to the file. This file is included in the component that is identified in the Driver Component setting. It is recommended that you have only one DLL per component; therefore, if you select a different DLL, delete the previous one from the Components view.

Table 12-52 • ODBC Driver Attributes (cont.)

Attribute	Expected Value
Setup Component	Enter the name of the component that you want to install the setup DLL for the ODBC driver. Usually, this component is the same one as the component for the driver if the setup code is in the same DLL.
Setup DLL	Enter the path to the setup DLL on the development system, or click the ellipsis button (...) to browse to the file. This file is included in the component that is identified in the Setup Component setting. If the driver file is also the setup DLL, you can leave this setting blank. It is recommended that you have only one DLL per component; therefore, if you select a different DLL, delete the previous one from the Components view.

Data Sources

In addition to the required attributes below, the DSN must have a name in the tree. This name is registered as the DSN's description. The DSN's name cannot be localized, which means that it does not appear in your project's list of string entries. Because of this, the same value is used regardless of the system's language.

Table 12-53 • Required Data Source Attributes

Attribute	Expected Value
Registration	Specify whether you want to use a system data source or a user data source. • **System data source**—The data source is available to all users on the system. • **User data source**—The data source is registered only for the current user.

Translators

In addition to the required attributes below, the translator must have a name in the tree. This name is registered as the translator's description. The translator's name cannot be localized, which means that it does not appear in your project's list of string entries. Because of this, the same value is used regardless of the system's language.

Table 12-54 • ODBC Translator Attributes

Attribute	Expected Value
Translator Component	Enter the name of the component that you want to install the translator and its attributes.
Translator DLL	Enter the path to the DLL file that is on the development system and that serves as the translator, or click the ellipsis button (...) to browse to the file. This file is included in the component that is identified in the Translator Component setting. It is recommended that you have only one DLL per component; therefore, if you select a different DLL, delete the previous one from the Components view.
Setup Component	Enter the name of the component that you want to install the setup DLL for the translator. Usually, this component is the same one as the component for the driver if the setup code is in the same DLL.

Table 12-54 • ODBC Translator Attributes (cont.)

Attribute	Expected Value
Setup DLL	Enter the path to the setup DLL on the development system, or click the ellipsis button (...) to browse to the file. This file is included in the component that is identified in the Setup Component setting. If the driver file is also the setup DLL, you can leave this setting blank. It is recommended that you have only one DLL per component; therefore, if you select a different DLL, delete the previous one from the Components view.

Note • *You cannot add an attribute to a translator.*

INI File Changes View

Project • *The INI File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

This view is not available in InstallScript projects; however, the InstallScript language includes .ini file functions for modifying .ini file data.

The INI File Changes view enables you to specify changes that should be made to initialization (.ini) files on the target system. These types of files serve as a database in which you can store and retrieve information between the uses of your applications. Some .ini files, such as **Boot.ini** and **Wininit.ini**, are used by the operating system. Although you can edit any .ini file found on the target system, editing system .ini files is not recommended.

Task ***Editing an .ini file involves three steps:***

1. Create an .ini file reference.
2. Add a section to the .ini file.
3. Add a keyword to the .ini file.

Before you can create an .ini file reference, you must have at least one component in your project. If no components exist when your .ini file reference is created, the Create a New Component dialog box is displayed, enabling you to create a component.

Tip *• InstallShield lets you import an existing .ini file into your project. Once you have imported the .ini file, you can use the INI File Changes view to configure the changes that you want to be made on the target system as needed. To learn more, see Importing an Existing .ini File.*

For details about each of the settings that you can configure for .ini file changes, see:

- .ini File Settings
- Section Settings for an .ini File
- Keyword Settings for an .ini File

.ini File Settings

Project *• The INI File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you add an .ini file to your project, configure the following settings:

Table 12-55 • .ini File Settings

Setting	Description
Display Name	Enter the name of the .ini file, including the file extension, that you would like to edit—for example, **INIFile.ini**. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. ***Note*** *• The display name is of the Windows Installer Filename data type. If the display name that you enter is not already in the 8.3 format, InstallShield uses the ShortName|LongName format for this setting. For example, if you enter* ***My File Name.ini*** *as the value in this setting, InstallShield sets the value to* ***MyFile~1.ini|My File Name.ini****, or something similar, so that a short name is available at run time if needed.*

Table 12-55 • .ini File Settings (cont.)

Setting	Description
Component	Select the component with which you want to associate this .ini file by using one of the following methods: • To select from a list of components that are already in your project, click the arrow. • To browse to the component that you want to use or to create a new component for the .ini file changes, click the ellipsis button (...). If the selected component is installed at run time, the .ini file is changed. The .ini file is not modified if the component is not installed.
Target	Specify the location of the .ini file on the target system. Instead of entering a hard-coded path, you can select a Windows Installer folder property from the list.  ***Tip*** • *Do not separate subfolders from the folder property with a backslash, but separate further levels of subfolders with a backslash—for example,* ***[ProgramFilesFolder]MyCompany\Subdirectory****.*

Section Settings for an .ini File



Project • *The INI File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you select a section in the INI File Changes view, the following setting is displayed.

Table 12-56 • .ini File Section Settings

Setting	Description
Display Name	Enter the section name of the .ini file that you would like to edit. Although the section name has square brackets around it in the .ini file, you do not need to include square brackets when you type the section name in this setting. For example, to make a change to the [INISection] section of an .ini file, you could type **INISection** in this setting. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Keyword Settings for an .ini File



Project • *The INI File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

When you select a keyword in the INI File Changes view, several settings are displayed.

Table 12-57 • .ini File Keyword Settings

Setting	Description
Display Name	Enter the name of the keyword that you would like to edit, exactly as it should appear in the target .ini file. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Action	Select the action that you would like to perform. Valid options are: • **Replace Old Value**—The existing value is replaced with the value that you enter in the Data Value setting. If it does not exist, the installation creates it. • **Do Not Overwrite**—Your value is added to the .ini file if the keyword does not already exist. No changes are made if the keyword is already present in the .ini file. • **Append Tag**—Select this option if you would like to add an additional tag to an keyword value. Tags are comma separated. If the keyword to which you would like to append a tag does not exist, no changes are made. • **Remove Whole Value**—The keyword and its value are both removed from the .ini file. If the specified keyword does not exist, no changes are made. If you select this option, leave the Data Value setting blank. • **Remove Tag**—Multiple values for a keyword are known as *tags*. Tags are separated by commas. To remove a tag from a keyword's value, select this option. In the Data Value setting, enter the tag that you want to be removed.
Data Value	Enter the value for the keyword. If you are adding or appending a value, enter the new value. If you are removing a tag, enter the tag that you would like to remove. If you are removing the keyword and value, leave this setting blank. You can use Windows Installer properties in your keyword's value. To do this, surround the property with square brackets—for example, **[INSTALLDIR]**.

Environment Variables View



Project *• The Environment Variables view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The InstallScript language includes the GetEnvVar function for retrieving the current value of an environment variable, and it enables you to create an environment variable.

Environment variables are variables that can be accessed by multiple applications on the target system. In the Environment Variables view, you can create new environment variables, modify the values of existing variables, and remove variables. The environment variable creation, modification, or removal takes place on the target system when the component that is associated with the environment variable is installed.

Environment Variable Settings



Project *• The Environment Variables view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The InstallScript language includes the GetEnvVar function for retrieving the current value of an environment variable, and it enables you to create an environment variable.

By specifying environment variable settings, you indicate how you want to affect existing variables on the target system and create new variables. Each environment variable property is described below:

Table 12-58 • Environment Variable Settings

Setting	Description
Component	Select the component with which you want to associate this environment variable by using one of the following methods: • To select from a list of components that are already in your project, click the arrow. • To browse to the component that you want to use or to create a new component for the environment variable, click the ellipsis button (...). The environment variable creation, modification, or removal takes place on the target system when the component that is associated with the environment variable is installed.
Value	Enter the path or value for this environment variable. You can use a predefined folder as part of this value—such as **[INSTALLDIR]Bin**. To enter multiple paths, separate the paths with a semicolon (;).  ***Note*** • *If you select Remove for the On Install setting, InstallShield clears any value entered in the Value setting, and the Value setting becomes read-only.*
On Install	Indicate the action that you want to take place when the associated component is installed to the target system. Available options are: • **Create**—The installation creates the specified environment variable on the target system—if it does not already exist—and sets its value. • **Set**—In conjunction with the Placement setting, this option sets the value of an existing environment variable. If you select this option and the environment variable does not already exist on the target system, the installation creates it and then sets its value. If the environment variable exists on the target system, the installation sets its value. • **Remove**—The installation removes the specified environment variable from the target system.  ***Note*** • *If you select Remove for the On Install setting, any value entered in the Value setting is cleared, and the Value setting becomes read-only.*

Table 12-58 • Environment Variable Settings (cont.)

Setting	Description
Placement	Select the action that you would like to perform. Available options are: • **Append**—The installation appends the new value that is entered in the Value setting to the end of the existing value. • **Prefix**—The installation adds the new value that is entered in the Value setting to the beginning of the existing value. • **Replace**—The installation replaces the value of the specified environment variable with the new value that is entered in the Value setting.  ***Note*** • *If you select Create for the On Install setting and the specified environment variable already exists on the target system, the Placement setting indicates how the new value should be added to the existing environment variable's value or if it should replace the existing environment variable's value. However, if—in this scenario—the specified environment variable does not exist on the target system, it is created and the Placement setting is ignored.*
On Uninstall	Indicate whether the environment variable should be updated when the associated component is uninstalled. Available options are: • **Remove**—If you select Append or Prefix for the Placement setting and the environment variable value on the target system contains the specified value at uninstallation time, only that value is removed from the existing variable's value. If you select Replace for the Placement setting, and if either of the following conditions are true at uninstallation time, the entire environment variable is removed: the value on the target system matches the specified value, or the value on the target system is empty. In all other cases, the environment variable and its value are left as is on the target system. • **Leave**—The environment variable and, if it is available, its value are left as is on the target system.
Type	Indicate whether the environment variable name refers to a system or user environment variable. Available options are: • **System**—The installation creates, modifies, or removes the specified system environment variable. • **User**—The installation creates, modifies, or removes the environment variable from the end user's environment. The specified environment variable is available only to the end user who is logged on at the time of installation.

XML File Changes View



Project • *The XML File Changes view is available in the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The XML File Changes view is where you add references for nodes and node sets of XML files that you want to change at run time. The XML files can be part of your installation, or they can be files that are already present on target systems.

Two key parts of this view are the View Filter (a list of the project's features and components) and the XML Files explorer (a tree view):

- The View Filter enables you to filter data by feature or component.
- The XML Files explorer is where you add references to XML files and XML data that you want to create, modify, or remove at run time. Each node represents an XPath expression that your installation carries out at run time to select existing nodes or node sets in an XML file.



Important • *The XML File Changes view is not designed to list a node for every node in an XML file. To improve performance, the XML File Changes view should show only the settings that differ from the base XML file:*

- *If the XML file that you are modifying is part of your installation, the XML File Changes view should list only the nodes and node sets that should be added, changed, or deleted after the XML file is installed at run time.*
- *If the XML file that you are modifying is a file that is already present on the target system, the XML File Changes view should list only the nodes that need to be added, changed, or deleted at run time.*

Therefore, when you are importing a file, import only the nodes in the XML file that you want to modify at run time.

Note that the XML File Changes view does not enable you to specify the order in which new elements should be listed in the XML file. Therefore, importing only the nodes that you want to modify at run time helps to avoid issues that may occur if your product requires that the elements be listed in a particular order.

The XML File Changes view also displays different tabs in the right pane, depending on what is selected in the XML Files explorer. For reference information about each of the settings on these tabs, see the following:

- General Tab for an XML File
- Advanced Tab for an XML File
- Namespace Tab for an XML File
- General Tab for an XML Element
- Advanced Tab for an XML Element

To learn more about how to use the XML Files Changes view and configure XML file changes, see Modifying XML Files.

General Tab for an XML File

Project • *The XML File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The General tab exposes the following settings for an XML file in the XML File Changes view:

Table 12-59 • General Tab Settings for a File in the XML File Changes View

Setting	Description
File Name	The file name represents the XML file name as it appears in the explorer. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) next to this setting to select an existing string. For more information, see Using String Entries in InstallShield.
XML File Destination	Specify the location of the XML file on the target machine. Click Browse to locate an existing directory or to add or create a new one.
Select the Features the XML File Belongs to	Select or clear the check boxes for one or more features that you want to associate with the XML file. InstallShield automatically associates a feature with an XML file when it is added to the XML File Changes view. ***Project*** • *For DIM and Merge Module projects—The Merged Feature check box in this pane is selected and disabled when you select an XML file in this view. When you add a DIM or merge module to an installation project, you specify the feature that should contain that DIM or merge module.*

Advanced Tab for an XML File

Project • *The XML File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

- *MSI Database*
- *Transform*

The Advanced tab exposes the following settings for an XML file in the XML File Changes view.

XML file options

The following file settings are available at run time:

Table 12-60 • XML File Settings Available at Run Time

Setting	Description
Always create XML file if it does not already exist	If an XML file does not exist on the target system at run time, then one will be created based on the specifications for that file in the XML File Changes view.
Remove XML file on uninstall	Select this check box if you want to remove the XML file on the target system upon uninstallation.
Format XML after changes	If you want the XML file to be formatted after the run-time changes are made to the file, select this check box. When formatting the file, the installation adds indentations to the file and replaces empty-element tags with start tags and end tags. Note that the formatting may cause problems for **web.config** files, so you may want to clear this check box for the XML file.
Encoding used for a new file	Specify the encoding that should be used for a new XML file. Note that if the XML file is already present on the target machine, or if it is being installed as part of your installation, the installation uses the encoding that is specified in the XML file, rather than the encoding that is specified in this setting. The encoding value that you specify is used only if you have used the XML File Changes view to create the entire XML file.

XPath Query

This read-only grid displays the XPath expressions to be executed on the target machine at run time based on the XML changes you have configured in the XML File Changes view.

Tip *• For more information about XPath expressions, consult the World Wide Web Consortium (W3C) Web site as well as the Things to Know and Avoid When Querying XML Documents with XPath article MSDN Library.*

Namespace Tab for an XML File

Project *• The XML File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The Namespace tab contains a table with the following settings for an XML file in the XML File Changes view.

Table 12-61 • XML Namespace Settings

Setting	Description
Prefix	Specify the prefix that should be added to elements that are associated with this namespace.
URI	Specify the uniform resource identifier that identifies the Internet resource for the namespace. The URI can be a URL or a string of characters. ***Note*** • *The MSXML parser does not use the URI to look up information about the namespace. The only purpose of the URI is to give the namespace a unique name within the XML file.*

General Tab for an XML Element

Project • *The XML File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The General tab exposes the following settings for an XML element in the XML File Changes view.

Element Name

The element name represents the declared element type for the XML file.

When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) next to this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Attributes

Table 12-62 • Settings for XML Element Attributes

Setting	Description
Attribute	Enter the name of an XML element attribute to created, updated, or removed at run time. You can add Windows Installer properties for the attribute. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Value	Enter the value for the attribute type. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Operation	Select from the following run-time operations: • **Create**—Adds the attribute to the element at run time. • **Remove**—Removes the attribute from the element at run time. • **Append**—Appends the attribute value that you specify to the end of the existing value.
Scheduling	Specify when to apply the attribute changes by selecting from the following scheduling options: Install, Uninstall, and Both.

Tip • *You can use dynamic values from Windows Installer properties or text substitution to store different values into the XML file.*

Advanced Tab for an XML Element

Project • *The XML File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The Advanced tab exposes the following settings for an XML element in the XML File Changes view.

Table 12-63 • XML Element Settings

Setting	Description
Update first matching element only	To modify only the first matching element in the target XML file, select this check box. If you clear this check box, every matching element in the target XML file is changed.
Always create this element if it does not already exist	To always create this element on the target machine if it does not already exist, select this check box. Note that if this check box is cleared for an element that is not present in the target file, its child elements are not created at run time. Thus, for an XML file such as //A/B/C, C is not created on the target system if B is neither present nor set to be created.
Remove element on uninstall	To remove this element from the XML file when the component containing this XML file change is uninstalled, select this check box.
Set Element Content	To add text content to the element, select this check box and enter the text in the Content box. In the following XML example, **feature** is the content for the element **product**: `<product>feature</product>`
Content	Enter a string with which to update the element. There is a 255-character limitation. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) next to this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Text File Changes View



Project • *The Text File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This view is not available in InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals.

The Text File Changes view is where you configure search-and-replace behavior for content in text files—for example, .txt, .htm, .xml, .config, .ini, and .sql files—that you want to modify at run time on the target system. The text files can be part of your installation, or they can be files that are already present on target systems.

Task ***Configuring text file changes involves the following steps:***

1. Create a text change set.
2. Specify the change for the text file.

Note *• Each text change set must be associated with a component in your project. Therefore, before you can create a text change set, your project must have at least one component. If no components exist when you are creating a text change reference, the Create a New Component dialog box is displayed, enabling you to create a component.*

Tip *• The Text File Changes view enables you to modify XML files without using MSXML (which is a run-time requirement if you use the XML File Changes view to modify XML files).*

For details about each of the settings that you can configure for text file changes, see:

- Change Set Settings
- Change Settings

Change Set Settings

Project *• The Text File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This view is not available in InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals.

When you select a change set item in the Text File Changes explorer of the Text File Changes view, the following settings are available for you to configure.

Table 12-64 • Change Set Settings

Setting	Description
Target Folder	Specify the location of the text file (that is, any non-binary file, such as a .txt, .htm, .xml, .config, .ini, or .sql file) on the target system. Instead of entering a hard-coded path, you can select a Windows Installer folder property from the list.  ***Tip*** • *Do not separate subfolders from the folder property with a backslash, but separate further levels of subfolders with a backslash—for example,* ***[ProgramFilesFolder]MyCompany\Subdirectory****.*
Include Files	Specify one or more text files that you want to be searched. Separate multiple files with a semicolon (;). To indicate a wild-card character, use an asterisk (*). For example, if you want to search all .xml and .config files in the specified directory, enter the following: ***.xml;*.config**  ***Tip*** • *You can use Windows Installer public properties to specify the names of the text files that you want to include in or exclude from your search. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's text files are modified at run time. To learn more, see Using Windows Installer Properties to Dynamically Modify Text Files.*
Exclude Files	Specify the file or files that you want to be excluded from the search. Separate multiple files with a semicolon (;). To indicate a wild-card character, use an asterisk (*). For example, if you want to exclude all .htm and .html files in the specified directory, enter the following: ***.htm;.html**  ***Tip*** • *You can use Windows Installer public properties to specify the names of the text files that you want to include in or exclude from your search. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's text files are modified at run time. To learn more, see Using Windows Installer Properties to Dynamically Modify Text Files.*
Include Subfolders	Specify whether you want the installation to search the subfolders of the location that you specified in the Target Folder setting.
Run on Install	Specify whether you want the search-and-replace behavior to occur when the change set's component is installed on the target system.

Table 12-64 • Change Set Settings (cont.)

Setting	Description
Run on Uninstall	Specify whether you want the search-and-replace behavior to occur when the change set's component is removed from the target system.
Component	Select the component with which you want to associate this text file by using one of the following methods: • To select from a list of components that are already in your project, click the arrow. • To browse to the component that you want to use or to create a new component for the text file changes, click the ellipsis button (...).

Change Settings



Project • *The Text File Changes view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

This view is not available in InstallScript projects; however, the InstallScript language includes string functions for finding and modifying string variables and literals.

When you select a change item in the Text File Changes explorer of the Text File Changes view, the following settings are available for you to configure.

Table 12-65 • Change Settings

Setting	Description
Action	Specify the type of change that you want to make in the applicable text file or files on target systems. Available options are: • **Replace**—Search for existing text and replace it. • **Insert at Beginning of File**—Add text to the beginning of the file. • **Insert at End of File**—Add text to the end of the file. • **Insert Before Text**—Search for existing text and add text before it. • **Insert After Text**—Search for existing text and add text after it.

Table 12-65 • Change Settings (cont.)

Setting	Description
Find What	This setting is available for the following types of text change actions: • Replace • Insert Before Text • Insert After Text Enter the string that you want to change, or select a Windows Installer property from the list. The list consists of all of the properties that are available in the Property Manager view of your project.  ***Tip*** • *You can use Windows Installer public properties to specify the search strings and the replacement strings. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's text files are modified at run time. To learn more, see Using Windows Installer Properties to Dynamically Modify Text Files.*
Replace With	This setting is available for the Replace type of text change action. Enter the new string that should replace the existing string that is found, or select a Windows Installer property from the list. The list consists of all of the properties that are available in the Property Manager view of your project.  ***Tip*** • *You can use Windows Installer public properties to specify the search strings and the replacement strings. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's text files are modified at run time. To learn more, see Using Windows Installer Properties to Dynamically Modify Text Files.*
Insert Text	This setting is available for the following types of text change actions: • Insert at Beginning of File • Insert at Beginning of File • Insert Before Text • Insert After Text Enter the new string that should be inserted into the text file or files.  ***Tip*** • *You can use Windows Installer public properties to specify the search strings and the inserted strings. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when your product's text files are modified at run time. To learn more, see Using Windows Installer Properties to Dynamically Modify Text Files.*

Table 12-65 • Change Settings (cont.)

Setting	Description
Match Whole Word Only	This setting is available for the following types of text change actions: • Replace • Insert Before Text • Insert After Text Specify whether you want to find only whole-word instances of the value that you have entered for the Find What setting. For example, if you specify Yes and you enter **install** for the Find What setting, the installation searches only for instances of *install*; it does not search for other forms of the word, such as *installs*, *installation*, or *uninstall*.
Match Case	This setting is available for the following types of text change actions: • Replace • Insert Before Text • Insert After Text Specify whether you want to restrict your search to strings with the same capitalization that you use in the Find What setting. For example, if you specify Yes and you enter **install** for the Find What setting, the installation searches only for all-lowercase instances of that string. If you select No, the installation searches for *install*, as well as *INSTALL*, *Install*, and other mixed-case instances.
Replace Only Once	This setting is available for the Replace type of text change action. Specify whether you want the installation to replace only the first occurrence of the search string. If you specify No, the installation replaces all instances of the search string.
Insert Only Once	This setting is available for the following types of text change actions: • Insert Before Text • Insert After Text Specify whether you want the installation to insert the text at only the first occurrence of the search string. If you specify No, the installation adds the string for all instances of the search string.

Scheduled Tasks View

Project • *The Scheduled Tasks view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The Scheduled Tasks view is where you add to your project automated tasks that you want to be created through the Windows task scheduler at run time on target systems. The scheduled tasks that you create can launch files that are part of your installation, or ones that are already present on target systems.

For details about each of the settings that you can configure for a scheduled task, see Scheduled Tasks Settings.

Note *• Each scheduled task must be associated with a component in your project. Therefore, before you can add a scheduled task, your project must have at least one component. If no components exist when you are adding a scheduled task, the Create a New Component dialog box is displayed, enabling you to create a component.*

Scheduled Tasks Settings

Project *• The Scheduled Tasks view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *Transform*

The settings for a scheduled task are organized into the following main categories in the Scheduled Task view.

- General
- Schedule

General

Use a scheduled task's General settings to specify details such as the name of the task, the component that contains the task, and the file that you want the task to run.

Table 12-66 • General Settings for a Scheduled Task

Setting	Description
Task Name	Enter the name that you want to use for the selected scheduled task. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) next to this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 12-66 • General Settings for a Scheduled Task (cont.)

Setting	Description
Component	Select the component that you want to contain this scheduled task by using one of the following methods: • To select from a list of components that are already in your project, click the arrow in this setting. • To browse to the component that you want to use or to create a new component for the scheduled task, click the ellipsis button (...) in this setting. If the installation installs this component at run time, the scheduled task is added to the target system.
Comments	Enter the description that you want to be displayed in the Windows task scheduler for this task. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) next to this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Target	Enter the path to the file on the target system that should be launched when the scheduled task is run. Use Windows Installer directory properties—for example, **[INSTALLDIR]File.exe**—instead of hard-coded directory paths. As an alternative to manually entering a value, you can click the ellipsis button (...) to browse to the target location and specify the file name.
Arguments	Enter the command-line arguments for the target file.
Working Directory	Enter the working directory for the target, or click the ellipsis button (...) to select or create a directory. For example, if you want your working directory to be set to a subdirectory of **INSTALLDIR** called **Files**, select **[INSTALLDIR]** from the list and add the subdirectory name **Files** to the end of it. When you are finished, it should read **[INSTALLDIR]Files**.
Run As	Enter the account that should be used to run the scheduled task. For example: **DomainName\UserName** Note that if you select Yes for this setting, you must also specify a password in the Password setting. Otherwise the scheduled task cannot be created at run time, and the installation aborts.  ***Tip*** • *You can use a Windows Installer public property to specify this information. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when a scheduled task is configured at run time. For more information, see Using Windows Installer Properties to Dynamically Configure a Scheduled Task.*

Table 12-66 • General Settings for a Scheduled Task (cont.)

Setting	Description
Password	Enter the password that should be used to with the user account that is specified in the Run As setting to run the scheduled task. Note that if you select Yes for the Run Only If Logged On setting, you must also specify a password in the Password setting. Otherwise the scheduled task cannot be created at run time, and the installation aborts.  ***Tip*** • *You can use a Windows Installer public property to specify this information. This enables you to use data that end users enter in dialogs, or other configuration information that is determined at run time, when a scheduled task is configured at run time. For more information, see Using Windows Installer Properties to Dynamically Configure a Scheduled Task.*
Run Only If Logged On	Specify whether you want the task to run only if the user account that is specified in the Run As setting is logged on to the system at the time that the task is scheduled. Note that if you select Yes for this setting, you must also specify a password in the Password setting (which is under the Run As setting). Otherwise the scheduled task cannot be created at run time, and the installation aborts.

Schedule

Use a scheduled task's Schedule settings to specify details such as how often you want the task to run. Note that some of the settings are displayed only if you select certain schedule options.

Table 12-67 • Schedule Settings for a Scheduled Task

Setting	Description
Schedule Type	Select the option that describes how often you want the scheduled task to be run.
Start Time	Enter the time that the task should start to be run in the following format: hh:mm where hh is the number of hours that have passed since midnight, and mm is the number of minutes that have passed since the start of the hour. The time that you specify should be from 00:00 to 23:59. For example, for a start time of 2:45 PM, enter **14:45**.
Start Date	Enter the date on which the task should start to be run. Use the same date format that is configured for the operating system of the machine on which you are using InstallShield.
Interval	Specify how often the scheduled task should be run, starting at the specified time and date.

Table 12-67 • Schedule Settings for a Scheduled Task (cont.)

Setting	Description
Run On	This setting varies, depending on the option that you select in the Schedule Type setting. • **For a weekly schedule**—To specify the days of the week on which you want the weekly scheduled task to be run, click the ellipsis button (...) in this setting, and then select the check boxes of the appropriate days. • **For a monthly schedule**—Specify whether you want the scheduled task to be run on a specific day or week of the month. • **For a run-once schedule**—Enter the date on which the task should be run. Use the same date format that is configured for the operating system of the machine on which you are using InstallShield.
Week	Select which week of the month you want the monthly scheduled task to be run.
Day	To specify the days on which you want the monthly scheduled task to be run, click the ellipsis button (...) in this setting, and then select the check boxes of the appropriate days.
Months	To specify the months of the year in which you want the monthly scheduled task to be run, click the ellipsis button (...) in this setting, and then select the appropriate check boxes.
Idle Time	Specify the number of minutes that the target system should be idle before running the scheduled task.

Services View



Project • *The Services view (and the Services area for a component) is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

You can use the Services view to configure a component that installs, starts, stops, or deletes a service during installation or uninstallation. You can also use this view to configure extended service customization options that are available with Windows Installer 5. As an alternative to using the Services view, you can use the Services area under the Advanced Settings node of the Components view or the Setup Design view. If you configure service information in any of these areas, the other areas are automatically updated accordingly.

Note the following details about working with services:

- The name that you enter for the service in the Services node must match the name that is shown on the service's Properties dialog box. (To access an installed service's properties: In the Services administrative tool, right-click the service and then click Properties.)
- The service that you are starting, stopping, deleting, or configuring can either be already present on the target system during installation or uninstallation, or it can be installed as part of your installation.
- The service executable file should be the key file of its component. For more information, see Component Key Files.
- The Remote Installation setting for the service's component must be set to Favor Local. For more information, see Setting a Component's Remote Installation Setting.



***Note** • A service must be a single executable file (.exe), since the Windows Installer does not support driver services.*

For details about each of the settings that you can configure for a service, see Services View Settings.

Services View Settings



***Project** • The Services view (and the Services area for a component) is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*



***Tip** • The View Filter at the top of the Services view lets you select a component or feature whose service data you want to display in the view. The View Filter lists your project's hierarchy of features, subfeatures, and components. Selecting a feature shows the services for all of the components in that feature.*

The Services settings are organized into the following main categories:

- General Settings
- Install Settings
- Control Settings
- Configure Settings



***Tip** • You must be familiar with the technical details of your service before you can configure its settings.*

General Settings

Use the General area to specify the component that you want to contain your service.

Table 12-68 • General Settings

Setting	Description
Component	Select the component that contains the service executable file, or click the ellipsis button (...) in this setting to create or select a component for the service executable file. The service executable file should be the key file of its component. For more information, see Component Key Files. In addition, the Remote Installation setting for the component must be set to Favor Local. For more information, see Setting a Component's Remote Installation Setting.

Install Settings

Use the Install Settings area to specify whether you want the installation to install your service. If the service should be installed, also use these settings to specify information such as the display name and description, as well as when the service should be started.

Table 12-69 • Install Settings

Setting	Description
Enable Service Install	Specify whether the service should be installed at run time. If you select No for this setting, any values that you entered for the remaining settings in the Install Settings section are deleted from the project. Note that when No is selected for this setting, the remaining settings in the Install Settings section are read-only.
Key Name	Enter a key name for the service. The key name is a database key; it is not displayed to end users.
Display Name	Enter the name that you want to be displayed in the service control manager for this service. If you leave this setting blank, the service's name (that is, the text that is used for the name of your service's subnode under the Advanced Settings node) is used. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 12-69 • Install Settings (cont.)

Setting	Description
Description	Enter a description of the service. This description is registered on the target system when the service is installed, and it is displayed in the service control manager's Description column. It is also displayed in the Description box on the General tab of the service's Properties dialog box. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Service Type	Select the type of service that you are installing. Available options are: • Win32 that runs in its own process • Win32 that shares a process The WIN32_OWN_PROCESS type of service contains the code for only one service. The WIN32_SHARE_PROCESS type of service contains code for more than one service, enabling them to share code.
Interact with Desktop	Specify whether the service interacts with the desktop. If your service has a user interface, select Yes. If you select Yes, the User Name setting must be left blank, since the service will be installed to run in the built-in LocalSystem account.
Start Type	Specify when to start the service. Available options are: • **Automatic**—The service starts automatically when the system starts. • **On Demand**—The service starts when the service is requested through the service control manager. • **Disabled**—The service cannot be started. Note that some services may support other start types (that is, during operating system initialization or by the operating system loader). However, these options are not available for the Start Type setting because the Windows Installer does not include support for them.
Error Control	Select the appropriate action that the service control manager should perform if the service fails to start. Available options are: • Log the error and continue • Log the error, display a message, and continue • Log the error and restart

Table 12-69 • Install Settings (cont.)

Setting	Description
Abort Install on Failure	Specify whether the entire installation should fail if the service cannot be installed on the target system. The default value is No. ***Note*** • *If you select Yes for this setting and end users run the installation in silent or basic UI mode, Windows Installer 3 or later must be present on the target system.*
Load Order Group	Enter the name of the load-ordering group, if any, of which this service is a member.
Dependencies	Enter any service or load-ordering groups that this service requires. The system attempts to start the required service or at least one member of the load-ordering group before starting this service. Separate multiple dependencies with a comma (,). You must precede the name of each load-ordering group with the SC_GROUP_IDENTIFIER—which is typically the plus sign (+)—so that the service control manager can distinguish it from a service.
User Name	Enter the account under which the service will be logged on. To install the service under the local system account, leave this setting blank. (Microsoft does not recommend installing services that impersonate the privileges of a single user.) If the service type is **Win32 that runs in its own process**, the value that you enter should use the following format: **DomainName\UserName** If the service will be logged on under the built-in domain, you can use the following format: **.\UserName**
Password	Enter a password for this service. Leave this setting blank if the User Name setting is empty—that is, when the service is logged on under the local system account. The password is not used if you have not specified a user name.
Start Parameters	Enter any command-line parameters or properties that are required to run the service.

Table 12-69 • Install Settings (cont.)

Setting	Description
Permissions	This setting lets you set permissions for your service. To configure permissions, click the Add New Service Permissions button. InstallShield adds a new permission setting, plus additional rows of related settings under it. Configure each of the settings as necessary. You may want to add multiple permission items if, for example, you want to pass different arguments during installation and uninstallation.  ***Note*** • *The permission settings are supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore these settings.* *An installation should not contain both the* ***MsiLockPermissionsEx*** *table and the* ***LockPermissions*** *table. If you use this Permissions setting, and the settings under it, you are configuring the* ***MsiLockPermissionsEx*** *table of the package. If* ***Traditional Windows Installer handling*** *is selected for the Locked-Down Permissions setting in the General Information view and you have set permissions for one or more files, folders, or registry keys in your project, you are configuring the* ***LockPermissions*** *table of the package.*
Key Name	Enter a key name for the permissions. The key name is an internal name; it is not displayed to end users.  ***Note*** • *The permission settings are supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore these settings.* *An installation should not contain both the* ***MsiLockPermissionsEx*** *table and the* ***LockPermissions*** *table. If you use this Permissions setting, and the settings under it, you are configuring the* ***MsiLockPermissionsEx*** *table of the package. If* ***Traditional Windows Installer handling*** *is selected for the Locked-Down Permissions setting in the General Information view and you have set permissions for one or more files, folders, or registry keys in your project, you are configuring the* ***LockPermissions*** *table of the package.*

Table 12-69 • Install Settings (cont.)

Setting	Description
Security Descriptor	Use valid security descriptor definition language (SDDL) to enter a valid SDDL string. For more information about SDDL, see the "Access Control" section of the Microsoft Windows Software Development Kit (SDK). ***Tip*** *• You can use angle brackets (for example,* ***<DomainName\UserName>****) or environment variables (for example,* ***[%UserDomain][%UserName]****) to indicate the domain and user name of the user whose account SID is going to be determined at run time.* ***Note*** *• The permission settings are supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore these settings.* *An installation should not contain both the* ***MsiLockPermissionsEx*** *table and the* ***LockPermissions*** *table. If you use this Permissions setting, and the settings under it, you are configuring the* ***MsiLockPermissionsEx*** *table of the package. If* ***Traditional Windows Installer handling*** *is selected for the Locked-Down Permissions setting in the General Information view and you have set permissions for one or more files, folders, or registry keys in your project, you are configuring the* ***LockPermissions*** *table of the package.*
Condition	This setting lets you enter a statement that the installation must evaluate before securing permissions for the service on the target system. The permissions are not configured if the condition evaluates to false. However, the permissions are configured if the condition evaluates to true, assuming that the service's feature is installed. To create a condition for the permissions, click the ellipsis button (...) in this setting. For more information, see Condition Builder Dialog Box. ***Note*** *• The permission settings are supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore these settings.* *An installation should not contain both the* ***MsiLockPermissionsEx*** *table and the* ***LockPermissions*** *table. If you use this Permissions setting, and the settings under it, you are configuring the* ***MsiLockPermissionsEx*** *table of the package. If* ***Traditional Windows Installer handling*** *is selected for the Locked-Down Permissions setting in the General Information view and you have set permissions for one or more files, folders, or registry keys in your project, you are configuring the* ***LockPermissions*** *table of the package.*

Control Settings

Use the Control Settings area to specify how you want to control the service when this component is installed or uninstalled. For example, before updating a running service, your installation may need to delete it.

Table 12-70 • Control Settings

Setting	Description
Events	This setting lets you specify one or more control events that are required for your service. To add an event, click the **Add New Service Control Event** button. InstallShield adds a new event setting, plus additional rows of related settings under it. Configure each of the settings as necessary. You can place all of your control options in a single event. However, you might need to create additional events if, for example, you want to pass different arguments during installation and uninstallation.
Key Name	Enter a key name for the control event. The key name is an internal name; it is not displayed to end users.
Operation Time	To specify how you would like to schedule the operation that occurs for the service at run time, select the appropriate option for each of the following settings: • **Install Start**—Specify whether you want to start the service during installation. If you select Yes, the service starts when the installation reaches the StartServices action in the Installation sequence. • **Install Stop**—Specify whether you want to stop the service during installation. If you select Yes, the service stops when the installation reaches the StopServices action in the Installation sequence. • **Install Delete**—Specify whether you want to delete the service during installation. If you select Yes, the service is deleted when the installation reaches the DeleteServices action in the Installation sequence. • **Uninstall Start**—Specify whether you want to start the service during uninstallation. If you select Yes, the service starts when the uninstallation reaches the StartServices action in the Installation sequence. • **Uninstall Stop**—Specify whether you want to stop the service during uninstallation. If you select Yes, the service stops when the uninstallation reaches the StopServices action in the Installation sequence. • **Uninstall Delete**—Specify whether you want to delete the service during uninstallation. If you select Yes, the service is deleted when the uninstallation reaches the DeleteServices action in the Installation sequence.

Table 12-70 • Control Settings (cont.)

Setting	Description
Wait Type	Specify how Windows Installer should proceed after initiating the control event. Available options are: • **Wait for the event to complete**—Windows Installer waits for the event—starting, stopping, or deleting—to finish, up to a maximum of 30 seconds, before proceeding. Select this option if the event is critical to the installation and you want to allow additional time for the event to return a failure error. • **Wait for the SCM**—Windows Installer waits for the service control manager to report that the service is in a pending state. The service control manager maintains the system's database of services and exposes an interface for controlling these services.
Service Argument	Specify the arguments that you want to be passed to the service. Separate multiple arguments with a comma (,).

Configure Settings

Use the Configure Settings area to specify how to customize the service that is included in the selected component.

Note • *The configuration settings are supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore these settings.*

Table 12-71 • Configure Settings

Setting	Description
Events	This setting lets you specify one or more configuration changes that are required for your service. To add an event, click the **Add New Service Configuration Event** button. InstallShield adds a new event setting, plus additional rows of related settings under it. Configure each of the settings as necessary. ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*
Key Name	Enter a key name for the configuration event. The key name is an internal name; it is not displayed to end users. ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*

Table 12-71 • Configure Settings (cont.)

Setting	Description
Run Time	To specify when you would like the service configuration event to occur at run time, select the appropriate option for each of the following settings: • **During Install**—Specify whether you want the event to occur during installation. • **During Uninstall**—Specify whether you want the event to occur during uninstallation. • **During Reinstall**—Specify whether you want the event to occur during reinstallation.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*

Table 12-71 • Configure Settings (cont.)

Setting	Description
Configuration Type	Specify what configuration change should be made to the service. The change takes effect the next time that the system is started. Available options are: • **Configure time delay of an auto-start**—This option is applicable if the service is an installed auto-start service, or if Automatic is selected for the Start Type setting in the Install Settings area. If you select this option, click the ellipsis button (...) in the Arguments setting to indicate the time delay. • **Configure when to run recovery actions**—If you select this option, click the ellipsis button (...) in the Arguments setting to specify when to run the recovery actions for the selected service. • **Add a service SID type to the process token**—If you select this option, click the ellipsis button (...) in the Arguments setting to indicate the appropriate security identifier (SID) type that you want to add for the service. • **Change list of required privileges**—If you select this option, click the ellipsis button (...) in the Arguments setting to select the appropriate privileges. • **Configure SCM pre-shutdown delay**—If you select this option, enter the length of the time delay (in milliseconds) in the Arguments setting. To reset the time delay to the default of 3 minutes, leave the Arguments setting blank. The service control manager waits for the specified period of time after sending the SERVICE_CONTROL_PRESHUTDOWN notification to the service. Note that each option has its own set of corresponding values for the Arguments setting. Therefore, if you change the value for the Arguments setting and next change the value for the Configuration Type setting, you may need to change the value for the Arguments setting again.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*
Arguments	To specify the value that corresponds with the configuration change that you selected in the Configuration Type setting, click the ellipsis button (...).  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*

Table 12-71 • Configure Settings (cont.)

Setting	Description
Recovery Actions	This setting lets you specify one or more recovery actions that are to be run after the service fails. To add a recovery action, click the **Add New Recovery Action** button. InstallShield adds a new recovery action setting, plus additional rows of related settings under it. Configure each of the settings as necessary.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*
Key Name	Enter a key name for the recovery action. The key name is an internal name; it is not displayed to end users.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*
Run Time	To specify when the system should run the recovery action after the service has failed, select the appropriate option for each of the following settings: • **During Install**—Run the action when the service's component is being installed. • **During Uninstall**—Run the action when the service's component is being uninstalled. • **During Reinstall**—Run the action when the service's component is being reinstalled.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*

Table 12-71 • Configure Settings (cont.)

Setting	Description
Reset Period	Specify the amount of time (in seconds) between the reset intervals for the service's failure count. The service control manager counts the number of times that the service has failed since the system was last restarted. When this interval has elapsed, the count is reset to the number 0 if the service has not failed during the reset period. When the service fails, the system performs an action that is specified for the SCM Actions setting. The appropriate action is determined by subtracting the number 1 from the number of failures since the last system restart. For example, if the action fails a sixth time, the system performs the fifth action type that is listed in the SCM Actions setting. If actions are not specified in the SCM Actions setting, the Reset Period setting is ignored. To indicate that the failure count should never be reset, leave this setting blank.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*
Reboot Message	Specify the message that should be displayed before the computer is restarted in response to a service control manager action of Reboot Computer. Note that Reboot Computer must be listed as one of the action types for the SCM Actions setting; otherwise, the Reboot Message setting is ignored. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*

Table 12-71 • Configure Settings (cont.)

Setting	Description
Run Command	Specify the command line that should be run in response to a service control manager action of Run Command. Use the absolute path; UNC paths are not supported. For example, enter **C:\myscripts\recovery.cmd**, rather than **\\computername\c$\myscripts\recovery.cmd**. Programs or scripts that you specify should not require input from end users. The command line that you specify is used by a new process that runs under the same account as the service. Note that Run Command must be listed as one of the action types for the SCM Actions setting; otherwise, the Run Command setting is ignored. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*
SCM Actions	This setting lets you specify one or more service control manager actions that should occur if the service fails. The actions are performed in the order that they are listed under this setting. To add an action, click the **Add New SCM Action Event** button. InstallShield adds a new Type setting, plus a Delay setting under it. Configure both of the settings as necessary.  ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*

Table 12-71 • Configure Settings (cont.)

Setting	Description
Type	Specify the type of action that the service control manager should perform if the service fails. Available options are: • Run Command • Reboot Computer • Take No Action • Restart Service ***Note*** • *The Restart Service action is not supported with .msi packages. Including this action with an .msi package will result in Windows Installer Error 1939.* ***Note*** • *The Type setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*
Delay	Specify the time (in milliseconds) that the service control manager should wait before performing the action that is specified in the Type setting. ***Note*** • *This setting is supported beginning with Windows Installer 5. Earlier versions of Windows Installer ignore this setting.*

Server Configuration View

Project • *The Server Configuration view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

InstallShield offers the following views to help you configure server-side installations.

Internet Information Services (IIS)



***Project** • The Internet Information Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

If you want to create an IIS Web site, application, virtual directory, application pool, or Web service extension on the target machine upon installation, you can configure it in the Internet Information Services view.

Component Services



***Project** • The Component Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The Component Services view enables you to manage COM+ server applications and components for your installation package.

SQL Scripts



***Project** • The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The SQL Scripts view provides a central location for managing and organizing all SQL scripts by connections and settings.

Internet Information Services View



***Project** • The Internet Information Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*
- *Merge Module*

The Internet Information Services view enables you to create and manage new IIS Web sites, applications, virtual directories, application pools, and Web service extensions.

The Internet Information Services view contains several areas:

- Web Sites (This area lets you add Web applications and virtual directories. For more information, see Application and Virtual Directory Settings.)
- Application Pools
- Web Service Extensions

The following table describes the buttons that are displayed above the settings in the Internet Information Services view. The buttons are displayed if a Web site, application, virtual directory, application pool, or Web service extension is selected in the center pane.

Table 12-72 • Controls in the Internet Information Services View

Name of Control	Icon	Description
Categorized		Sorts the settings according to categories.
Alphabetical		Sorts the settings alphabetically.



Tip • *You can configure an IIS setting dynamically at run time through the use of Windows Installer public properties (in Basic MSI, DIM, InstallScript MSI, and Merge Module projects) or text substitution string variables (in InstallScript projects). This enables you to let end users specify the name of the virtual directory, the TCP port, the site number, or other IIS settings for the Web sites, applications, virtual directories, application pools, and Web service extensions that they are installing on the target machine. To learn more, see Using Windows Installer Properties to Dynamically Modify IIS Settings and Using InstallScript Text Substitution to Dynamically Modify IIS Settings.*

Web Site Settings



Project • *The Internet Information Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Use the Web Sites item in the Internet Information Services view to add and delete Web sites, and to configure system-wide settings for the Web server.

Web Server Settings that Apply to All Web Sites in a Project

When you select the Web Sites explorer in the Internet Information Services view, the following Web server settings are displayed.

Table 12-73 • Web Server Settings

Setting	Description
Restart Web Server After Configuring IIS (IIS 6 and earlier only)	To restart IIS after each time the installation is done making IIS changes to the system, select Yes. Some applications may require IIS to be restarted. This setting applies to IIS 6 and earlier. IIS 7 ignores this setting.
SSIEnableCmdDirective Registry Value	Specify how you want the SSIEnableCmdDirective registry value for the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters registry key to be set on the target system. The SSIEnableCmdDirective registry value controls whether the Web server allows the #exec CMD directive of server-side includes (SSIs) to be used to execute shell commands. Valid options are: • **Ignore**—Do not change the SSIEnableCmdDirective registry value on the target system. This is the default option. • **FALSE (0)**—Set the SSIEnableCmdDirective registry value on the target system to 0. This prevents the #exec CMD directive of server-side includes to be used to execute shell commands. Note that if you select this value and an IIS Web server has applications that rely on #exec CMD directives, those applications may stop working properly after your installation project's Web site and virtual directory are installed. • **TRUE (1)**—Set the SSIEnableCmdDirective registry value on the target system to 1. This allows the #exec CMD directive of server-side includes to be used to execute shell commands. If you select the FALSE or TRUE options, InstallShield stores the value—either 0 for FALSE or 1 for TRUE—in the **INSTALLSHIELD_SSI_PROP** property. Because of security concerns, the default SSIEnableCmdDirective registry value is FALSE (0); the FALSE (0) value prevents end users from running unauthorized server-side executable files.  ***Note*** • *If your product is uninstalled from a target system, the SSIEnableCmdDirective registry value is not changed, even if its value was changed during installation.* For more information, see Specifying Whether a Web Server Should Allow the CMD Command to Be Used for SSI #exec Directives.



Note • *The aforementioned Web server settings are not updated on a target system at installation run time if no IIS items (Web sites, applications, virtual directories, application pools, or Web service extensions) are installed.*

Settings that Are Configurable for Each Web Site in a Project

When you select a Web site in the explorer, many settings are displayed. The Web site settings are organized into several main categories:

- Identification
- General
- Home Directory
- Application Settings
- Security
- Advanced

Identification Settings

The following settings are available in the Identification area for a Web site in the Internet Information Services view.

Table 12-74 • Identification Settings for a Web Site

Setting	Description
Name	Enter a name for the Web site. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
IP Address	To target a specific IP address, enter it. As an alternative, you can leave an asterisk (*), which is the default value. An asterisk or a blank value for this setting indicates that any unused IP address should be used.
TCP Port	The TCP Port setting for an IIS Web site indicates the port number on which the service is running on the target machine. Some versions of the IIS Web server can host multiple Web sites. Each Web site is associated with a unique port number. If you do not know what the port number on the target system should be, you can enter 0 for this setting. To learn which port and site numbers are used for the Web site when it is installed, see Configuring the TCP Port and Site Numbers.
Host Header Name	To specify the host header name that identifies the IIS Web site that is installed during your installation, type it in this box. For example: **www.mycompany.com** Host headers (also known as domain names) enable you to assign more than one Web site to an IP address on a Web server.

Table 12-74 • Identification Settings for a Web Site (cont.)

Setting	Description
Site Number	The Site Number setting indicates the number in the path at which the Web site will be created (that is, w3svc/3). The default value is 0. If you do not know what the site number on the target system should be, you can enter 0 for this setting. To learn which port and site numbers are used for the Web site when it is installed, see Configuring the TCP Port and Site Numbers.

General Settings

The following settings are available in the General area for a Web site in the Internet Information Services view.

Table 12-75 • General Settings for a Web Site

Setting	Description
Component	Select the component with which the Web site is associated. You can also click the ellipsis button (...) to locate an existing component or create a new one.
ASP.NET Version	To set the ASP.NET version for the Web site, enter the complete version number, or select it from the list. For example, to specify version 2 of ASP.NET, type **2.0.50727**. To specify version 1.1 of ASP.NET, type **1.1.4322**. If you specify an ASP.NET version for a Web site, IIS uses that same version number for any of the Web site's applications.  ***Important*** • *If your installation may be run on a Windows Vista or later system, you may not want to set the ASP.NET version. Also note that if you specify version 3 of ASP.NET, an error occurs at run time. For more information, see Setting the ASP.NET Version for a Web Site or Application.*
ASP.NET Platform	If the installation may be run on a 64-bit version of Windows with the .NET Framework, specify which ASP.NET platform should be used to map the Web site or application to the ASP.NET version: • **32-bit**—The 32-bit version of the ASP.NET IIS Registration tool should be used. Select this option if Yes is selected for the application pool's Enable 32-Bit Applications setting; otherwise the installation fails. • **64-bit**—The 64-bit version of the ASP.NET IIS Registration tool should be used. Select this option if No is selected for the application pool's Enable 32-Bit Applications setting; otherwise the installation fails. For more information, see Considerations for Supporting IIS 6 on 64-Bit Platforms.

Table 12-75 • General Settings for a Web Site (cont.)

Setting	Description
Delete on Uninstall	Specify whether you want to delete the selected Web site during uninstallation. For more information, see Uninstalling Web Sites, Applications, and Virtual Directories.
Default Documents	Type the name of the default page on your Web site. To specify multiple pages, separate each with a comma. A Web site serves a default page whenever a browser request does not specify a document name.

Home Directory Settings

The following settings are available in the Home Directory area for a Web site in the Internet Information Services view.

Table 12-76 • Home Directory Settings for a Web Site

Setting	Description
Content Source Path (Local or UNC)	This setting identifies the local path or network directory path that stores your Web site files. • If the content for the Web site is on the target system, click the ellipsis button (...) in this setting to specify a local path. The Browse for Directory dialog box opens. In a Basic MSI or InstallScript MSI project, this dialog box enables you to select a Windows Installer property (such as [IISROOTFOLDER]) or create a new one. In an InstallScript project, this dialog box enables you to select an InstallScript variable (such as <IISROOTFOLDER>) or create a new one. By default, the files are stored in IISROOTFOLDER. • If the content for the Web site is not on the target system, click the UNC button in this setting to specify a network location. Following is an example: \\server\share  ***Tip*** • *Each Web site should have a unique physical path, especially if the Web site is going to be installed on a Windows Vista or later system or a Windows Server 2008 or later system. To learn more, see Run-Time Requirements for IIS Support.*
Script Source Access	Specify whether you want to allow end users to access source code if either read or write permissions are set. Source code includes scripts in ASP applications.
Read Access	Specify whether you want to provide end users with read access to the Web site.
Write Access	Specify whether you want to provide end users with write access to the Web site. This means that end users can change the Web site's properties on the target machine.

Table 12-76 • Home Directory Settings for a Web Site (cont.)

Setting	Description
Directory Browsing	Specify whether you want to allow end users to see all the virtual directories and subdirectories below this Web site.
Log Visits	Specify whether you want to record visits to this Web site in a log file. Visits are recorded only if logging is enabled for this Web site.
Index this Resource	Specify whether you want to allow Microsoft Indexing Service to include this directory in a full-text index of your Web site. This setting applies to IIS 6 and earlier. IIS 7 ignores this setting.

Application Settings

The following settings are available in the Application Settings area for a Web site in the Internet Information Services view.

Table 12-77 • Application Settings for a Web Site

Setting	Description
Application Pool	To associate the selected Web site with an application pool, select its name in the list. As an alternative, you can click the ellipsis button (...) to select or create a string entry for the application pool. For more information, see Using String Entries in InstallShield. This setting applies to IIS 6 and later. Earlier versions of IIS ignore this setting.  ***Important*** • *The application pool that you specify should be on the target system at run time or part of your installation; otherwise, the server may generate an error such as error 403.18.*
Application Mappings	To customize the directory's application mappings, click the ellipsis button (...) in this setting. This opens the Application Mappings dialog box, which enables you to add, edit, and delete a mapping between a file name extension and the application that processes those files.
MIME Types	To add, edit, or delete MIME types for the selected Web site, click the ellipsis button (...) in this setting. This opens the MIME Types dialog box, which enables you to add, edit, and delete a mapping between a file name extension and the corresponding content type that is served as a static file by the Web server on the target system to a browser or mail client.
Session Timeout (minutes)	Specify the number of minutes that a session can remain idle before the server terminates it automatically. If the end user does not refresh or request a page within the timeout period, the session ends. The default value is 20 minutes.

Table 12-77 • Application Settings for a Web Site (cont.)

Setting	Description
ASP Script Timeout (seconds)	Specify the length of time in seconds that .asp pages will allow a script to run before terminating and writing an event to the Windows Event Log. The minimum value for this property is 1 second; the default value is 90 seconds.

Security Settings

When you select a Web site in the Internet Information Services view, InstallShield displays several security-related settings in the Security area. The Security area lets you configure your Web server to verify the identify of users. You can authenticate users to prevent unauthorized ones from establishing a Web (HTTP) connection to restricted content. For more information, refer to the IIS documentation.

The Security area contains several categories of settings:

- Anonymous Connection
- Authenticated Access
- Secure Communication

The settings in the Anonymous Connection area are as follows.

Table 12-78 • Anonymous Connection Settings in the Security Settings Area

Setting	Description
Enable Anonymous Access	Specify whether you want to allow users to establish an anonymous connection. If you do want to allow anonymous connections, also enter the appropriate Windows user account information. If you do not need the Web server to confirm the identity of end users before they can access the content, select No for this setting.
IIS Controls Anonymous Password	Specify whether you want to automatically synchronize your anonymous password settings with those set in Windows on the target system. If the password that you type for the anonymous account differs from the password that Windows has, anonymous authentication will not work.  ***Note*** • *Password synchronization should be used only with anonymous user accounts defined on the local computer, not with anonymous accounts on remote computers.*
Anonymous User Name	If you are enabling anonymous connections, type the name of the anonymous account.
Anonymous Password	If you have selected No for the IIS Controls Anonymous Password setting, type the anonymous user account password. The password is used only within Windows. Anonymous users do not log on by using a user name and password.

The settings in the Authenticated Access area are as follows.

Table 12-79 • Authenticated Access Settings in the Security Settings Area

Setting	Description
Basic authentication	Specify whether you want to enable the basic authentication method for collecting user name and password information for end users who access the Web site.  ***Important*** • *With the basic method of authentication, user names and passwords are not encrypted when they are transmitted across the network. An unscrupulous end user who has network monitoring tools could intercept user names and passwords.*
Integrated Windows authentication	Specify whether you want to enable integrated Windows authentication. Integrated Windows authentication uses a cryptographic exchange with the end user's browser to confirm the identity of the user. When integrated Windows authentication is enabled, the Web site will use it only under the following conditions: • Anonymous access is disabled. • Anonymous access is denied because Windows file system permissions have been set, requiring end users to provide a Windows user name and password before establishing a connection with restricted content.

The settings in the Secure Communication area are as follows.

Table 12-80 • Secure Communication Settings in the Security Settings Area

Setting	Description
SSL certificate	To specify a server certificate that should be installed on the target system, click the ellipsis button (...) in this setting, and then select the appropriate security certificate file (.cer or .pfx). As an alternative, you can select a certificate from the list if your project already contains one or more certificates. InstallShield stores the .cer file in the **Binary** table. If no certificate is configured to be installed, this setting is blank.
SSL certificate password	If the certificate that you specified has a password, enter it in this setting.

Advanced Settings

The following settings are available in the Advanced area for a Web site in the Internet Information Services view.

Table 12-81 • Advanced Settings for a Web Site

Setting	Description
Custom Errors	To customize HTTP errors that are sent to clients when Web server errors occur, select the ellipsis button (...) in this setting. The Custom Errors dialog box opens, enabling you to specify the page that should be displayed for one or more HTTP errors. Administrators can use generic HTTP 1.1 errors, detailed custom error pages that IIS provides, or your own custom error pages that are you including in the installation.
Other IIS Properties	To specify values for IIS settings that are not displayed in the other areas in this view, select the ellipsis button (...) in this setting. The Other IIS Properties dialog box opens, enabling you to set the value of one or more IIS properties. To learn more, see Configuring Advanced IIS Settings. The advanced settings apply to IIS 6 and earlier. IIS 7 ignores these settings. For help on specific settings, see Metabase Property Reference on the MSDN Web site.

Application and Virtual Directory Settings



Project • *The Internet Information Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

You can add applications and virtual directories to your Web site in the Internet Information Services view. When you select an application or virtual directory in this view, many settings are displayed. The settings are organized into several main categories:

- General
- Virtual Directory
- Application Settings
- Security
- Advanced

General Settings

The following settings are available in the General area for an application or a virtual directory in the Internet Information Services view.

Table 12-82 • General Settings for an Application or a Virtual Directory

Setting	Description
Name	Enter a name for the application or virtual directory. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Component	Select the component with which the application or virtual directory is associated. You can also click the ellipsis button (...) to locate an existing component or create a new one.
ASP.NET Version	 ***Note*** • *This setting applies to applications but not virtual directories.* To set the ASP.NET version for the application, enter the complete version number, or select it from the list. For example, to specify version 2 of ASP.NET, type **2.0.50727**. To specify version 1.1 of ASP.NET, type **1.1.4322**. If you specify an ASP.NET version for a Web site, IIS uses that same version number for any of the Web site's applications.  ***Important*** • *If your installation may be run on a Windows Vista or later system, you may not want to set the ASP.NET version. Also note that if you specify version 3 of ASP.NET, an error occurs at run time. For more information, see Setting the ASP.NET Version for a Web Site or Application.*

Table 12-82 • General Settings for an Application or a Virtual Directory (cont.)

Setting	Description
ASP.NET Platform	 ***Note*** • *This setting applies to applications but not virtual directories.* If the installation may be run on a 64-bit version of Windows with the .NET Framework, specify which ASP.NET platform should be used to map the application to the ASP.NET version: • **32-bit**—The 32-bit version of the ASP.NET IIS Registration tool should be used. Select this option if Yes is selected for the application pool's Enable 32-Bit Applications setting; otherwise the installation fails. • **64-bit**—The 64-bit version of the ASP.NET IIS Registration tool should be used. Select this option if No is selected for the application pool's Enable 32-Bit Applications setting; otherwise the installation fails. For more information, see Considerations for Supporting IIS 6 on 64-Bit Platforms.
Default Documents	Type the name of the default page on your application or virtual directory. To specify multiple pages, separate each with a comma. A application or virtual directory serves a default page whenever a browser request does not specify a document name.

Virtual Directory Settings

The following settings are available in the Virtual Directory area for an application or a virtual directory in the Internet Information Services view.

Table 12-83 • Virtual Directory Settings for an Application or a Virtual Directory

Setting	Description
Content Source Path (Local or UNC)	This setting identifies the local path or network directory path that stores the default files for your application or virtual directory. • If the content for the application or virtual directory is on the target system, click the ellipsis button (...) in this setting to specify a local path. The Browse for Directory dialog box opens. In a Basic MSI or InstallScript MSI project, this dialog box enables you to select a Windows Installer property (such as [IISROOTFOLDER]) or create a new one. In an InstallScript project, this dialog box enables you to select an InstallScript variable (such as <IISROOTFOLDER>) or create a new one. By default, the files are stored in IISROOTFOLDER. • If the content for the application or virtual directory is not on the target system, click the UNC button in this setting to specify a network location. Following is an example: \\server\share  ***Tip*** • *Each application or virtual directory should have a unique physical path, especially if it is going to be installed on a Windows Vista or later system or a Windows Server 2008 or later system. To learn more, see Run-Time Requirements for IIS Support.*
Script Source Access	Specify whether you want to allow end users to access source code if either read or write permissions are set. Source code includes scripts in ASP applications.
Read Access	Specify whether you want to provide end users with read access to the application or virtual directory.
Write Access	Specify whether you want to provide end users with write access to the application or virtual directory. This means that end users can change the application's or virtual directory's properties on the target machine.
Directory Browsing	Specify whether you want to allow end users to see all the virtual directories and subdirectories below this application or virtual directory.
Log Visits	Specify whether you want to record visits to this application or virtual directory in a log file. Visits are recorded only if logging is enabled.

Table 12-83 • Virtual Directory Settings for an Application or a Virtual Directory (cont.)

Setting	Description
Index this Resource	Specify whether you want to allow Microsoft Indexing Service to include this application or virtual directory in a full-text index. This setting applies to IIS 6 and earlier. IIS 7 ignores this setting.

Application Settings

The following settings are available in the Application Settings area for an application or a virtual directory in the Internet Information Services view.

Table 12-84 • Application Settings for an Application or a Virtual Directory

Setting	Description
Application Name	To associate the selected virtual directory with an application, specify the application name. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Note*** • *This setting is available for virtual directories, but not for applications.* *If the virtual directory was created in InstallShield 2009 or earlier and then upgraded to the current version of InstallShield, this setting is displayed. Otherwise, this setting is not included.*
Application Pool	 ***Note*** • *This setting applies to applications but not virtual directories.* To associate the selected application with an application pool, select its name in the list. As an alternative, you can click the ellipsis button (...) to select or create a string entry for the application pool. For more information, see Using String Entries in InstallShield. This setting applies to IIS 6 and later. Earlier versions of IIS ignore this setting.  ***Important*** • *The application pool that you specify should be on the target system at run time or part of your installation; otherwise, the server may generate an error such as error 403.18.*

Table 12-84 • Application Settings for an Application or a Virtual Directory (cont.)

Setting	Description
Application Mappings	To customize the directory's application mappings, click the ellipsis button (...) in this setting. This opens the Application Mappings dialog box, which enables you to add, edit, and delete a mapping between a file name extension and the application that processes those files.
MIME Types	To add, edit, or delete MIME types for the selected application or virtual directory, click the ellipsis button (...) in this setting. This opens the MIME Types dialog box, which enables you to add, edit, and delete a mapping between a file name extension and the corresponding content type that is served as a static file by the Web server on the target system to a browser or mail client.
Session Timeout (minutes)	Specify the number of minutes that a session can remain idle before the server terminates it automatically. If the end user does not refresh or request a page within the timeout period, the session ends. The default value is 20 minutes.
ASP Script Timeout (seconds)	Specify the length of time in seconds that .asp pages will allow a script to run before terminating and writing an event to the Windows Event Log. The minimum value for this property is 1 second and the default value is 90 seconds.
Execute Permissions	Specify what level of program execution is allowed for the selected application or virtual directory. Available options are: • **None**—Only static files such as HTML and image files can be accessed. • **Scripts Only**—Only scripts such as ASP scripts can be run. • **Scripts and Executables**—All file types can be accessed or run.
Application Protection	***Note*** • *This setting applies to applications but not virtual directories.* Specify the level of protection: • **High**—The application is run in an isolated process that is separate from other processes. • **Medium**—The application is run in an isolated pooled process with other applications. • **Low**—The application is run in the same process as Web services. This setting applies to IIS 5 and earlier. Later versions ignore this setting.

Security Settings

When you select an application or a virtual directory in the Internet Information Services view, InstallShield displays several security-related settings in the Security area. The Security area lets you configure your application or virtual directory to verify the identify of users. You can authenticate users to prevent unauthorized ones from establishing a Web (HTTP) connection to restricted content. For more information, refer to the IIS documentation.

The Security area contains the following categories of settings:

- Anonymous Connection
- Authenticated Access

The settings in the Anonymous Connection area are as follows.

Table 12-85 • Anonymous Connection Settings in the Security Settings Area

Setting	Description
Enable Anonymous Access	Specify whether you want to allow users to establish an anonymous connection. If you do want to allow anonymous connections, also enter the appropriate Windows user account information. If you do not need the Web server to confirm the identity of end users before they can access the content, select No for this setting.
IIS Controls Anonymous Password	Specify whether you want to automatically synchronize your anonymous password settings with those set in Windows on the target system. If the password that you type for the anonymous account differs from the password that Windows has, anonymous authentication will not work.  ***Note*** • *Password synchronization should be used only with anonymous user accounts defined on the local computer, not with anonymous accounts on remote computers.*
Anonymous User Name	If you are enabling anonymous connections, type the name of the anonymous account.
Anonymous Password	If you have selected No for the IIS Controls Anonymous Password setting, type the anonymous user account password. The password is used only within Windows. Anonymous users do not log on by using a user name and password.

The settings in the Authenticated Access area are as follows.

Table 12-86 • Authenticated Access Settings in the Security Settings Area

Setting	Description
Basic authentication	Specify whether you want to enable the basic authentication method for collecting user name and password information for end users who access the application or virtual directory.  ***Important*** • *With the basic method of authentication, user names and passwords are not encrypted when they are transmitted across the network. An unscrupulous end user who has network monitoring tools could intercept user names and passwords.*

Table 12-86 • Authenticated Access Settings in the Security Settings Area (cont.)

Setting	Description
Integrated Windows authentication	Specify whether you want to enable integrated Windows authentication. Integrated Windows authentication uses a cryptographic exchange with the end user's browser to confirm the identity of the user. When integrated Windows authentication is enabled, the Web site will use it only under the following conditions: • Anonymous access is disabled. • Anonymous access is denied because Windows file system permissions have been set, requiring end users to provide a Windows user name and password before establishing a connection with restricted content.

Advanced Settings

The following settings are available in the Advanced area for an application or a virtual directory in the Internet Information Services view.

Table 12-87 • Advanced Settings for an Application or a Virtual Directory

Setting	Description
Custom Errors	To customize HTTP errors that are sent to clients when Web server errors occur, select the ellipsis button (...) in this setting. The Custom Errors dialog box opens, enabling you to specify the page that should be displayed for one or more HTTP errors. Administrators can use generic HTTP 1.1 errors, detailed custom error pages that IIS provides, or your own custom error pages that are you including in the installation.
Other IIS Properties	To specify values for IIS settings that are not displayed in the other areas in this view, select the ellipsis button (...) in this setting. The Other IIS Properties dialog box opens, enabling you to set the value of one or more IIS properties. To learn more, see Configuring Advanced IIS Settings. The advanced settings apply to IIS 6 and earlier. IIS 7 ignores these settings. For help on specific settings, see Metabase Property Reference on the MSDN Web site.

Application Pool Settings



Project • *The Internet Information Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*
- *Merge Module*



***Note** • Application pools are available only on machines with IIS 6.0 or later installed.*

Use the Application Pools item in the Internet Information Services view to add and delete application pools.

When you select an application pool in the explorer, many settings are displayed. The application pool settings are organized into several main categories:

- General
- CPU Settings
- Process Model
- Recycling

General Settings

The following settings are available in the General area for an application pool in the Internet Information Services view.

Table 12-88 • General Settings for an Application Pool

Setting	Description
Name	Provide a display name for the application pool that you want to configure. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Component	Select the component with which the application pool is associated. You can also click the ellipsis button (...) to locate an existing component or create a new one.
Overwrite Existing Application Pool	Select the appropriate option to indicate the behavior that you want to occur if the application pool already exists on the target system at run time. Available options are: • **Yes**—If the application pool already exists on the target system, the installation overwrites its settings with the values that you have configured for it in your project. During uninstallation, the application pool is uninstalled. • **No**—If the application pool already exists on the target system, the installation does not overwrite any of its settings; during uninstallation, the application pool is left on the target system. If the application pool does not exist on the target system, the installation creates it; during uninstallation, the application pool is uninstalled.

Table 12-88 • General Settings for an Application Pool (cont.)

Setting	Description
Mark Component as Permanent	Specify whether you want the component that contains the application pool to be removed when the component's feature is uninstalled. If you want the component to be marked as permanent so that it is not uninstalled, select Yes.
.NET Framework Version	If appropriate, select the .NET Framework version that the application pool should load. This setting applies to IIS 7 and later. Earlier versions of IIS ignore this setting.
Queue Length	Specify the maximum number of requests that HTTP.sys should queue for the application pool. The default value is 4000.
Managed Pipeline Mode	Specify the appropriate request-processing pipeline mode for the selected application pool: • **Integrated**—The ASP.NET request processing is integrated into the IIS 7 request-processing pipeline. This is the default option. • **Classic**—IIS routes requests for code through ISAPI, which processes the requests as if the application were running in IIS 6. This setting applies to IIS 7. Earlier versions of IIS ignore this setting.
Enable 32-Bit Applications	Specify whether you want to allow 32-bit applications in the selected application pool to be run on 64-bit systems. This setting applies to application pools that are installed on IIS 7 or later. Earlier versions of IIS ignore this setting. For more information, see Considerations for Supporting IIS 6 on 64-Bit Platforms.

CPU Settings

The following settings are available in the CPU area for an application pool in the Internet Information Services view.

Table 12-89 • CPU Settings for an Application Pool

Setting	Description
Limit	Specify the maximum percentage of CPU time (in 1/1000ths of a percent) that the worker processes in the application pool are allowed to consume over the period of time that is indicated for the Limit Interval setting. To avoid limiting the worker processes to a percentage of CPU time, specify the number 0. If you leave this setting blank, InstallShield does not configure the CPU limit, and whatever value is set by default on the server is used for your application pool.

Table 12-89 • CPU Settings for an Application Pool (cont.)

Setting	Description
Limit Action	Specify the action that should occur if the CPU limit that is set for the application pool is exceeded. Available options are: • **No action**—An event entry is added to the event log. No other action occurs. • **Shutdown**—An event entry is added to the event log. In addition, the application pool is shut down for the remainder of the limit interval.
Limit Interval (minutes)	Specify the amount of time (in minutes) between the reset intervals for CPU monitoring and throttling limits on the application pool. When this time interval has elapsed, IIS resets the CPU timers for the logging and limit intervals. To disable CPU monitoring, specify the number 0. If you leave this setting blank, InstallShield does not configure the CPU limit interval, and whatever value is set by default on the server is used for your application pool.

Process Model Settings

The following settings are available in the Process Model area for an application pool in the Internet Information Services view.

Table 12-90 • Process Model Settings for an Application Pool

Setting	Description
Identity	Specify the account under which the application pool's worker process should run. Available options are: • **NetworkService**—This account is a member of the Users group, and it has the fewest user rights that are required to run the application. This account provides the most security against an attack that might try to take over the Web server. • **LocalService**—This account is a member of the Users group, and it has the same user rights as the NetworkService account. However, the LocalService account is limited to the local computer. • **LocalSystem**—This account has all user rights, and it is part of the Administrators group. • **SpecificUser**—If you do not want to use one of the predefined accounts, select this option, and also specify the user name and password for the account that you want to use. • **ApplicationPoolIdentity**—This account uses a virtual identity that is unique to the selected application pool for running the application pool's worker process. Support for this option is available on target systems that have IIS 7 and later. If this new option is selected in an installation that is run on a system that has IIS 6, the NetworkService account is used instead for the application pool's identity.  ***Note*** • *If you use a SpecificUser identity, ensure that the user account that you specify is a member of the IIS_IUSRS group on the Web server so that the account can access the appropriate resources.*
SpecificUser User Name	If you select SpecificUser for the Identity setting, enter the specific user's user name.
SpecificUser User Password	If you select SpecificUser for the Identity setting, enter the specific user's password.  ***Project*** • *In Basic MSI and InstallScript MSI projects, you can specify a Windows Installer property for the password. To prevent the password from being logged, set the value of the* ***MsiHiddenProperties*** *property to the name of the Windows Installer property that you are using for the password. For more information, see MsiHiddenProperties in the Windows Installer Help Library.*

Table 12-90 • Process Model Settings for an Application Pool (cont.)

Setting	Description
Idle Timeout (minutes)	Specify the amount of time (in minutes) for which a worker process should remain idle before it is shut down.
Maximum Worker Processes	Specify the maximum number of worker processes that can service requests for the application pool. If you specify a number greater than 1, the application pool is considered to be a Web garden.

Recycling Settings

The Recycling settings for an application pool in the Internet Information Services view let you configure the recycling of worker processes. In worker process isolation mode, you can configure IIS to periodically restart worker processes in an application pool, enabling you to manage precisely those worker processes that are faulty. This helps to ensure that specified applications in those pools remain healthy and that system resources can be recovered.

The following settings are available in the Recycling area for an application pool in the Internet Information Services view.

Table 12-91 • Recycling Settings for an Application Pool

Setting	Description
Regular Time Interval (minutes)	Specify the amount of time (in minutes) after which the application pool should be recycles. To prevent the application pool from being recycled on a regular interval, enter the number 0.
Request Limit	Specify the maximum number of requests that the application pool can process before it is recycled. To allow the application pool to process an unlimited number of requests, enter the number 0.
Specific Times	To specify the time of day when the application pool should be recycled, click the ellipsis button (...) in this setting. The Recycling Specific Times dialog box opens, enabling you to configure the specific local time or times. When you specify a time, use the 24-hour format. For example, to specify 10:00 P.M., enter a time of **22:00**. In this setting, InstallShield indicates the number of times that were specified, and also lists the times in square brackets. Times are separated with a vertical bar. For example, the following value indicates that the specific recycling times are 5:00 A.M. and 11:00 P.M.: **2 defined [05:00|23:00]**

Web Service Extension Settings

Project • *The Internet Information Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Note • *Web service extensions are available only on machines with IIS 6.0 or later installed.*

On systems with IIS 7, Web service extensions require that the IIS Metabase and IIS 6 Configuration Compatibility feature be installed. For more information, see Version-Specific Information for IIS Support in InstallShield.

Use the Web Service Extensions item in the Internet Information Services view to add and delete Web service extensions.

When you select a Web service extension in the explorer, the following settings are available for you to configure.

Table 12-92 • General Settings for a Web Service Extension

Setting	Description
Description	Type the name of the new Web service extension file. The name is associated with a string ID, which can be translated. Click the Browse button to locate an existing string, or create a new one once the Select String dialog box has been launched.
Component	Select the component with which the Web service extension is associated. You can also click the ellipsis button (...) to locate an existing component or create a new one.
Mark Component as Permanent	Specify whether you want the component that contains the Web service extension to be removed when the component's feature is uninstalled. If you want the component to be marked as permanent so that it is not uninstalled, select Yes.
Full Path to File	Type the name of the Web service extension file. ***Tip*** • *You can use directory IDs to specify a file. For example, you can specify a directory and a file as* ***[INSTALLDIR]file.exe****.*
Group ID	Type the name of Web service extension's group ID. Group IDs enable you to classify different DLLs and common gateway interfaces (CGIs) and have dependencies for the applications. ***Tip*** • *You can use directory IDs to specify a group.*

Table 12-92 • General Settings for a Web Service Extension (cont.)

Setting	Description
Allow	Specify whether you want to set the status of the Web service extension to Allow.
UI Deletable	Specify whether you want to allow the Web service extension file to be deleted from within IIS Manager.
Overwrite Existing Extensions	Specify whether you want to enable existing extensions to be overwritten.

Component Services View



Project • *The Component Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The Component Services view enables you to manage COM+ applications and components for your installation package. You can manage both COM+ server applications and application proxies. A COM+ application proxy consists of a subset of the attributes of the server application, and it enables remote access from a client machine to the machine where the application resides.

Note the following information regarding component services in InstallShield:

- Only non-system COM+ applications can be added to your project. Therefore, InstallShield displays only non-system COM+ applications under the COM+ Applications explorer in the Component Services view.
- Only the COM+ applications that are installed on the local machine are included in the Component Services view and available for you to add to your projects.
- An application proxy is available for COM+ server applications only, not for library applications.

When you select a COM+ application in the Component Services view, several tabs are displayed:

- Installation
- General
- Security
- Identity
- Activation
- Queuing
- Advanced
- Dump
- Pooling/Recycling

The settings on the Installation tab are InstallShield-specific settings. The settings on the other tabs are similar to those in the Component Services administrative tool in the Control Panel. For details on each of the settings on the Installation tab, see Installation Tab. To learn about the settings on the other tabs, consult the Component Services help.

Installation Tab



Project • *The Component Services view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The Installation tab is one of the tabs that is displayed when you select a COM+ application in the Component Services view.

Table 12-93 • Installation Tab Settings

Setting	Description
Server	To install the selected COM+ application on other machines, select this check box.
Destination (Server)	The default destination location for COM+ applications is: **[ProgramFilesFolder]COMPlus Applications\{UID}** To install the COM+ files to a different location, select the target destination. If the destination that you want to specify is not available in the list, select the **Browse, create, or modify a directory entry** option.
Condition (Server)	Specify a condition for the COM+ server application if appropriate.
Install user identities with roles	To install the selected COM+ application with the user identities and roles that are configured for the COM+ application on your local machine, select this check box.
Refresh the COM+ settings from the client machine at build	The Component Services view shows COM+ settings that are available in Component Services on the local machine. To refresh the COM+ settings that are displayed in your project with the settings that are available from Component Services on the local machine, select this check box. InstallShield refreshes the settings each time that you build a release.

Table 12-93 • Installation Tab Settings (cont.)

Setting	Description
Install after InstallFinalize action	If the selected COM+ application contains .NET assemblies that need to be installed to the global assembly cache (GAC), select this check box. If you select this check box, the ISComponentServiceFinalize action installs the selected COM+ application after the InstallFinalize action. Windows Installer does not commit changes made in the in-script session to the GAC until InstallFinalize.
Proxy	To install the selected COM+ application as an application proxy, select this check box. A COM+ application proxy consists of a subset of the attributes of the server application, and it enables remote access from a client machine to the machine where the application resides.
Destination (Proxy)	The default destination location for COM+ applications is: **[ProgramFilesFolder]COMPlus Applications\{UID}** To install the COM+ files to a different location, select the target destination. If the destination that you want to specify is not available in the list, select the **Browse, create, or modify a directory entry** option.
Condition (Proxy)	Specify a condition for the proxy server support if appropriate.
Remote Server Name	Specify the name of the remote server computer where the application resides. You can type the exact name or use the default **[REMOTESERVERNAME]** property, which is automatically created when you select the Proxy check box for a COM+ application in your installation.  ***Note*** • *The default value for the **[REMOTESERVERNAME]** property is the name of the machine used to add the COM+ application to the installation project in InstallShield. To change the value of the [**REMOTESERVERNAME**] property, use the Property Manager view.* If you would like the end user to be able to specify the remote server, add a Remote Server edit field control to an end-user dialog in the Dialogs view. Set the Property value of this control to **REMOTESERVERNAME**.

Table 12-93 • Installation Tab Settings (cont.)

Setting	Description
Enable distributed COM on the client machine	Select this check box if appropriate. Clear this check box if you know that distributed component object model (DCOM) is already enabled on all client machines and you will not have administrative privileges on them. If you select this check box, Y is written at installation time to the EnableDCOM entry of the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Ole registry key to enable DCOM. ***Note*** *• End users can enable or disable DCOM on their machines using the Component Services administrative tool in the Control Panel. However, the application proxy will not work on a client machine if DCOM is disabled on that machine. For this reason, you may want to select the* ***Enable distributed COM on the client machine*** *check box.* *If an end user uninstalls the application proxy support, the EnableDCOM registry entry is not changed, even if the installation process involved changing this registry entry to Y on the target machine.*
Features	Select the feature that should contain the selected COM+ application. To add the COM+ application to a new feature, first create a feature in the Features view, and then select its check box in this list.

SQL Scripts View

Project *• The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The SQL Scripts view provides a central location for managing and organizing all SQL scripts by server connections and settings. This view provides support for Microsoft SQL Server, Microsoft Windows Azure, MySQL, and Oracle. You can perform most of the following major functions to configure your SQL servers from the SQL Scripts view. Some limitations apply to certain server types.

- Connect to SQL servers.
- Import catalog schema and/or data.
- Associate SQL scripts with features.
- Set required SQL server/script properties (server name, database name, authentication method, etc.).
- Set SQL script for execution during installation or uninstallation.

- Edit SQL scripts.
- Require and/or target specific versions of Windows Azure SQL, SQL Server, MySQL, or Oracle.
- Define SQL script text replacement.
- Open scripts in Microsoft SQL Server Management Studio or Microsoft SQL Server Query Analyzer.



***Note** • The import database functionality applies to the Microsoft SQL Server Database. Oracle users should refer to the Oracle Web page on Oracle Database Utilities for information on utilities that may work in conjunction with InstallShield.*

If you have Microsoft SQL Server Management Studio or Microsoft SQL Server 2000 SQL Query Analyzer installed on your system, you can open a new SQL script that you have added to your project to test, edit, and syntax-check the script. To launch one of those tools and open your script from within InstallShield, right-click the script file in the SQL Scripts view and then click Open Script in Microsoft SQL Server Management Studio. InstallShield searches your system for the following tools in order and launches the first one that it finds:

1. *Microsoft SQL Server 2008 Management Studio (any edition, including Express;* **ssms.exe***)*
2. *Microsoft SQL Server 2005 Management Studio (***SqlWb.exe***)*
3. *Microsoft SQL Server 2005 Management Studio Express (***ssmsee.exe***)*
4. *Microsoft SQL Server 2000 Query Analyzer (***isqlw.exe***)*

SQL Connections

In the SQL Scripts view, scripts are organized by connection, since no script can run on a server until a connection has been established. Furthermore, the grouping of connections with corresponding scripts facilitates the sharing of connection settings to different scripts.

When you create or select a SQL connection in the SQL Scripts view, the following tabs are available:

- General tab
- Requirements tab
- Advanced tab

General Tab



***Project** • The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

You can configure the following settings on the General tab for a SQL connection in the SQL Scripts view.

Table 12-94 • General Tab Settings for a SQL Connection

Setting	Description
Catalog Name	 ***Note*** • *Creating a new application catalog based on Oracle database settings is equivalent to creating a new database based on SQL Server settings in InstallShield. However, terminology is slightly different in reference to Oracle. In Oracle, the catalog is equivalent to the schema being used for every new application catalog. Therefore, note the semantic differences between "catalog" and "database" when understanding the differences between SQL Server and Oracle support in the documentation.* Enter the name of the SQL catalog to which you want to create a connection during the installation. If you leave this setting blank, the installation attempts to connect to the default catalog during the connection test phase at run time. If the catalog you specify is not found at that time, the installation still creates the specified catalog upon connection verification. This is because the Create Catalog If Absent option is selected in InstallShield by default.

Create Catalog If Absent	This option is selected by default. If this option is selected, the installation creates the specified catalog upon connection verification at the SQLLogin dialog at run time.
	
	You should note that when the **Create Catalog If Absent** option is selected, different SQL commands are issued for the various supported server types.
	
	For Microsoft SQL Server, Microsoft Windows Azure SQL, and MySQL, selecting this option results in the following:
	
	`CREATE DATABASE CatalogName`
	
	CatalogName is the value that you specify in the Catalog Name box on the General tab for a connection in the SQL Scripts view.
	
	For Oracle, selecting the **Create Catalog If Absent** option results in the following:
	
	`CREATE USER CatalogName IDENTIFIED BY CatalogName DEFAULT TABLESPACE USERS QUOTA UNLIMITED on USERSGRANT CONNECT TO CatalogNameGRANT DBA TO CatalogNameALTER USER CatalogName DEFAULT ROLE ALL`
	
	Therefore, you should note that for Oracle, the schema name serves as the user name and password for the selected catalog by default.

Table 12-94 • General Tab Settings for a SQL Connection (cont.)

Setting	Description
Create Catalog If Absent (cont.)	If you do not wish to use the **Create Catalog If Absent** option to create a catalog, you can run a customized script to create a catalog. When you run a custom script, you can use the Runtime tab for the script in the SQL Scripts view to schedule the execution of the script during login. For more information about running a customized script, see the following instructions: • Creating a Sample Installation that Creates a SQL Server Database by Running Customized SQL Script • Creating a Sample Installation that Creates an Oracle Schema by Running Customized SQL Script  ***Project*** • *In the case where the* ***Create Catalog If Absent*** *option is cleared during installation design and the catalog is not found at run time, the installation behavior slightly differs for the following project types at run time.* • *For Basic MSI, DIM, and InstallScript MSI projects—The installation connects to the default catalog at the SQLLogin dialog during run time to test the connection. This enables the installation to run custom code during the InstallExecuteSequence. However, the presence of the catalog needs to be guaranteed when the SQL scripts that are specified in your project are executed. The run time re-establishes the connection with the specified catalog to run the scripts. If the catalog is absent at this point, the connection fails.* • *For InstallScript projects—The catalog needs to be present at the SQLLogin dialog since the run time holds the connection until the installation ends, once the connection is established. In this scenario, you can run custom InstallScript code to create the catalog before the SQLLogin dialog shows up to override the default behavior.*

Table 12-94 • General Tab Settings for a SQL Connection (cont.)

Setting	Description

Default Target Server Name	For Microsoft SQL Server and MySQL, you can enter the machine name of the target SQL Server here. This setting is optional.
	
	For Microsoft Windows Azure SQL, enter a fully qualified server name in the following format:
	
	`tcp:`*`ServerName`*`.database.windows.net`
	
	Following is an example:
	
	`tcp:wbzdh64drd.database.windows.net`
	
	For Oracle client machines with Oracle 11g client software installed, enter a SQL connection URL string in the following format:
	
	`//host: [port] [/service name]`
	
	The following is an example of what you would type to connect to a specified remote Oracle machine:
	
	`//sch01jsmithrxp.installshield.com:1521/ORCL`
	
	You can also specify a local net service name if you have tnsnames.ora configured on the client machine. For more details, consult the Oracle Support Web site.

Connect using	Select the type of authentication you want to use to connect to the specified catalog. For SQL Server Authentication, enter login and password information for the targeted server.
Comments	Enter comments about this connection. These comments are not displayed to the end user.

Requirements Tab



Project • *The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

On the Requirements tab, you can select a database server and specify version requirements for the type of server that the installation should target at run time. You have the option to add, delete, or edit existing version requirements that are associated with a particular database server.

Task ***To select a database server and specify version requirements for the type of server that the installation will target at run time:***

1. In the **Database Servers** list, select the check box for the appropriate database server, and then highlight that server type.

   If you select more than one database server, at run time, the installation establishes a connection and performs the specified requirement checks on the selected server types based on the sequence of the server types as they appear in the table. That is, if the installation finds a match after checking requirements against the first database server in the table, a connection and a check is not done on any selected subsequent database types in the table. The SQL scripts that are associated with the connection are installed to that first database server that is verified.

2. Click the **Add** button. The New Version Requirement dialog box for the type of database server that you highlighted opens.
3. Specify version requirements for the type of server that the installation should target at run time.
4. Click **OK**.

InstallShield adds a new database server entry and its appropriate version information to the table next to the Database Servers list.

Task ***To delete or edit an entry in the Database Servers table:***

1. Select the database server that the installation is targeting at run time.
2. Select the version requirement that you want to delete or edit.
3. Click **Delete** or **Edit**. When you click **Edit**, either the Edit Version Requirement dialog box for the appropriate database server type opens.
4. Update the settings and click **OK**.

Note • *Multiselection of items in the version requirements section of the table applies only to the Delete operation.*

Allow installation to continue when minimum requirements are not met

If you want to allow the installation to continue if the minimum database server requirements are not met, select this check box.

If you select this check box and the minimum requirements are not met, the installation skips the SQL connection and all of its SQL scripts, and continues with the rest of the installation.

If you clear this check box and the minimum requirements are not met, the installation does not allow the end user to continue with the rest of the installation.

This check box is cleared by default.

Allow installation to Microsoft SQL Server Desktop Engine/SQL Server Express

Select this check box if it is true. Clear this check box if you want to prohibit installation to SQL Server Desktop Engine and SQL Server Express.

Advanced Tab



Project • *The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The Advanced tab provides advanced settings that should be set only by expert users to expand and customize the default functionality that is available in InstallShield.

Table 12-95 • Advanced Tab Settings for a SQL Connection

Setting	Project Type	Description
Command Timeout (seconds)	Basic MSI, DIM, InstallScript, InstallScript MSI	Specify the command timeout period. The default is 30 seconds. The valid range of this value is 0 to 2147483647. A value of 0 indicates no limit. Once this period of time has elapsed without completing the command, an error occurs and ADO cancels the command.

Table 12-95 • Advanced Tab Settings for a SQL Connection (cont.)

Setting	Project Type	Description

Batch Separator	Basic MSI, DIM, InstallScript, InstallScript MSI	Specify the appropriate batch separator for the selected connection. The batch separator is used for all SQL scripts that are associated with this connection.
		
		The default value is **GO**, which is the same default that is used in Microsoft products. Oracle utilities use a slash (**/**) as the default value.
		
		A batch separator is a command that is recognized by SQL utilities such as osql, isql, and Microsoft SQL Server Management Studio, as well as InstallShield. The run time interprets a batch separator as a signal that it should send the current batch of SQL statements to a database server.
		
		
		
		Note • *A semicolon (;) is the only batch separator that does not need to be on a separate new line by itself. All other batch separators must be on a separate line. For example, if you specify a slash as a batch separator and you have the following SQL script, the installation sends* ***some statement*** *to a SQL server as a batch first; then it sends* ***another statement*** *as another batch:*
		
		`some statement`
		`/`
		`another statement`
		`/`
		
		If you have the following script, the installation sends all of the lines, including the slashes at the end of lines, as a batch:
		
		`some statement/`
		`another statement/`

Table 12-95 • Advanced Tab Settings for a SQL Connection (cont.)

Setting	Project Type	Description
Target Server Property Name	Basic MSI, DIM, InstallScript MSI	Select the property that identifies one of the following, depending on the type of database server that the installation is targeting: • The name of the target server instance (for Microsoft SQL Server and MySQL) • The fully qualified server name (for Microsoft Windows Azure SQL) • The connect URL string or local net service name (for Oracle) The default property is **IS_SQLSERVER_SERVER**. If your project contains more than one SQL connection and you want to be able to use different properties for each connection, you can do so. For more information, see Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties.
Target Catalog Property Name	Basic MSI, DIM, InstallScript MSI	This setting enables you to select a property that identifies the name of the SQL catalog to which you want to create a connection during the installation. The default property is **IS_SQLSERVER_DATABASE**. If your project contains more than one SQL connection and you want to be able to use different properties for each connection, you can do so. For more information, see Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties.
Authentication Type Property Name	Basic MSI, DIM, InstallScript MSI	This setting enables you to select a property that identifies the type of authentication that you want to use to connect to the specified catalog. The default property is **IS_SQLSERVER_AUTHENTICATION**. The following numbers are valid property values: • **0**—Windows authentication credential of the current user • **1**—Server authentication If your project contains more than one SQL connection and you want to be able to use different properties for each connection, you can do so. For more information, see Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties.

Table 12-95 • Advanced Tab Settings for a SQL Connection (cont.)

Setting	Project Type	Description
Server Authentication Login ID Property Name	Basic MSI, DIM, InstallScript MSI	This setting enables you to select a property that identifies the login ID that should be used for server authentication. The default property is **IS_SQLSERVER_USERNAME**. If your project contains more than one SQL connection and you want to be able to use different properties for each connection, you can do so. For more information, see Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties.
Server Authentication Password Property Name	Basic MSI, DIM, InstallScript MSI	This setting enables you to select a property that identifies the password that should be used for server authentication. The default property is **IS_SQLSERVER_PASSWORD**. If your project contains more than one SQL connection and you want to be able to use different properties for each connection, you can do so. For more information, see Specifying Whether New SQL Connections Should Share the Same Windows Installer Properties.

Project • *If you change any of the SQL properties on the Advanced tab of a SQL connection in a Basic MSI project, the corresponding properties are not automatically updated in the SQLLogin dialog in the Dialogs view. Therefore, you must manually change the properties in the dialog to match the properties that selected on the Advanced tab of the SQL connection.*

SQL Script Level

Project • *The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

When you click a SQL script in the SQL Scripts view, the following tabs are available:

- General tab
- Script tab
- Runtime tab
- Database Import tab
- Text Replacement tab

Note • *If you have Microsoft SQL Server Management Studio or Microsoft SQL Server 2000 SQL Query Analyzer installed on your system, you can open a new SQL script that you have added to your project to test, edit, and syntax-check the script. To launch one of those tools and open your script from within InstallShield, right-click the script file in the SQL Scripts view and then click Open Script in Microsoft SQL Server Management Studio. InstallShield searches your system for the following tools in order and launches the first one that it finds:*

1. *Microsoft SQL Server 2008 Management Studio (any edition, including Express;* **ssms.exe***)*
2. *Microsoft SQL Server 2005 Management Studio (***SqlWb.exe***)*
3. *Microsoft SQL Server 2005 Management Studio Express (***ssmsee.exe***)*
4. *Microsoft SQL Server 2000 Query Analyzer (***isqlw.exe***)*

General Tab

Project • *The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

You can configure the following settings on the General tab for a SQL script in the SQL Scripts view.

Table 12-96 • General Tab Settings for a SQL Script

Setting	Description
SQL Script File Name	Click the ellipsis button (...) next to this setting to browse to the SQL script that your installation will execute at run time.
Select Features the SQL Script Belongs to	Select or clear the check boxes for one or more features that you want to contain the SQL script. ***Project*** • *For DIM projects—The Merged Feature check box in this pane is selected and disabled when you select a SQL script in this view. When you add a DIM to an installation project, you specify the feature that should contain that DIM.*
Component Name	This setting shows the name of the component that contains the SQL script. To move to the Components view and see the component that contains this SQL script, click the hyperlink.

Table 12-96 • General Tab Settings for a SQL Script (cont.)

Setting	Description
Schema Version	Specify a version number to enable script versioning. For more information on this setting, see Specifying a Version Number for a SQL Script File. ***Tip*** *• When you specify a number for the Schema Version setting and InstallShield adds the custom* ***InstallShield*** *table for storing the schema version number on the target database, the data is not automatically removed upon uninstallation. Therefore, if you want the installation to be able to roll back the changes, you need to create a custom script upon uninstallation to drop the* ***InstallShield*** *table.*
Comments	Enter comments about this SQL script. These comments are not displayed to the end user.

Script Tab

Project *• The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The Script tab is intended for advanced users. To create a script from an existing database with step-by-step instructions, use the Database Import Wizard.

Note *• This functionality applies to the Microsoft SQL Server Database.*

Runtime Tab

Project *• The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

You can configure the following settings on the Runtime tab for a SQL script in the SQL Scripts view.

Table 12-97 • Runtime Tab Settings for a SQL Script

Setting	Project Type	Description
Script Execution	Basic MSI, DIM, InstallScript, InstallScript MSI	To indicate when you want the installation to execute a SQL script, select the appropriate check box. Available options are: • **Run Script During Install**—This option enables you to run a custom script during installation. This option is also associated with component state because each script is associated in with a feature. Therefore, in Basic MSI, DIM, and InstallScript MSI projects, you can specify conditional statements in conjunction with this setting in the Script Condition area of this tab. In Basic MSI, DIM, and InstallScript MSI projects, the ISSQLServerInstall action is associated with this option. In InstallScript projects, the InstallScript function **SQLRTComponentInstall** is associated with this option. • **Run Script During Uninstall**—This option enables you to run a custom script during uninstallation. This option is also associated with component state because each script is tied in with a feature by design. Therefore, you can specify conditional statements in conjunction with this setting in the Script Condition (only available for Basic MSI and InstallScript MSI projects) section of this tab. In Basic MSI, DIM, and InstallScript MSI projects, the **ISSQLServerUninstall** action is associated with this option. In InstallScript projects, the InstallScript function **SQLRTComponentUninstall** is associated with this option.

Table 12-97 • Runtime Tab Settings for a SQL Script (cont.)

Setting	Project Type	Description
Script Execution (cont.)		• **Run Script During Rollback**—This option enables you to run a custom script during rollback. InstallShield does not roll back the changes automatically. You will have to run a custom script allowing rollback when you select the Run Script During Rollback option. In Basic MSI, DIM, and InstallScript MSI projects, the ISSQLServerRollback action is associated with this option. • **Run Script During Login**—This option enables you to run your script during login. Note that there are some limitations with this option. For example, note that the script changes are irreversible once the end user clicks the Next button on the SQLLogin dialog at run time. If the end user attempts to cancel the installation before clicking the Install button on the ReadyToInstall dialog, the script changes are not rolled back. Also note that the script is executed after credentials and requirements are verified and before a connection has been made to the catalog. Therefore, any schema version requirements that you set for the script during installation design will not have taken place at run time. In Basic MSI, and DIM projects, the ISSQLServerValidate action is associated with this option. In InstallScript MSI projects, the InstallScript function **SQLRTServerValidate** is associated with this option. In InstallScript projects, the InstallScript function **SQLRTConnect** is associated with this option.
Script Error Handling	Basic MSI, DIM, InstallScript, InstallScript MSI	Specify how you want SQL script errors to be handled at run time. Available options are: • On Error, Go to Next Script • On Error, Go to Next Statement • On Error, Abort Installation
Script Condition	Basic MSI, DIM, InstallScript MSI	If you want to specify a condition that Windows Installer must evaluate before the SQL script is run during installation or uninstallation, select the check box and type the conditional statement in the text box. The SQL script is not run if the condition evaluates to false. As an alternative to manually entering a statement, you can click the ellipsis button (...) to launch the Condition Builder dialog box, which simplifies the process of creating a condition.

Database Import Tab



***Project** • The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*



***Tip** • The Database Import functionality currently is only available for Microsoft SQL Server. Oracle users should refer to the Oracle Web page on Oracle Database Utilities for information on utilities that may work in conjunction with InstallShield.*

You can run the Database Import Wizard from this tab to view and modify your import settings. The wizard will walk you through the process of generating a SQL script that re-creates part or all of an existing Microsoft SQL Database. Once you complete the wizard, you can click the Regenerate button on this tab to refresh your script.

The **Regenerate SQL Script at Build** option lets you indicate whether the script should be regenerated each time that you build a release. Regenerating a script every time you build your release can slow down the build process.

Text Replacement Tab



***Project** • The SQL Scripts view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The Text Replacement tab lets you specify strings to replace in your SQL script at run time. You should replace text only if it is unique and will not cause any script syntax errors. At run time, the installation replaces the text according to the parameters that you specify.

To add a new search-and-replace entry, click the Add button. To modify an existing search-and-replace entry, select that entry and then click the Edit button. In both cases, the Find and Replace dialog box opens, enabling you to specify search-and-replace parameters.

To delete an existing search-and-replace entry, select that entry and then click the Delete button.

Behavior and Logic View



***Project** • The Behavior and Logic view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

By adding custom actions; sequencing actions and dialogs; adding support files; using InstallScript; searching the target system for required files, folders, or other elements; or configuring Windows Installer properties, you can design any custom functionality that your installation requires.

InstallScript



Project *• The InstallScript view is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

InstallScript is an installation authoring language that is similar to the C programming language. The InstallScript view provides you with a script editor pane in which you can create and edit your scripts.

Custom Actions and Sequences



Project *• The Custom Actions and Sequences view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Sequences direct all the actions that are performed during installation, from file transfer to displaying the user interface. Use the Custom Actions and Sequences view to sequence the actions and dialogs in your project.

If Windows Installer does not directly support functionality that is required by your installation program, you can extend your installation program through the use of custom actions in the Custom Actions and Sequences view.

Custom Actions



Project *• The Custom Actions view is available in the following project types:*

- *DIM*
- *Merge Module*
- *MSM Database*

Although merge modules do support the use of custom actions, sequences are not defined through a dedicated view in merge module projects.



***Tip** • You can control the launch of custom actions in a merge module by modifying the **ModuleInstallExecuteSequence** table in the Direct Editor. When you add the merge module to your installation project, all custom actions and dialogs included in the merge module are available for you to insert in the installation's sequences via the Custom Actions and Sequences view.*

Support Files/Billboards



***Project** • The Support Files view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript Object*
- *Suite/Advanced UI*

The Support Files/Billboards view is available in the following project types:

- *InstallScript*
- *InstallScript MSI*

Support Files

The Support Files view lets you add, sort, and delete support files—files that are required by your installation project *only* during the installation process.

Billboards

The Billboards area in the Support Files/Billboards view lets you add billboards to your project. Billboards are images that are displayed for a specified amount of time while your installation runs. You can use billboards to provide information about the product being installed or to entertain the end user during the installation process.

System Search



***Project** • The System Search view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*

- *Transform*

Use the System Search view to search for a particular file, folder, registry key, or .ini value on the target system prior to an installation.

Property Manager



Project *• The Property Manager view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

In Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, and Transform projects, the Property Manager view enables you to edit the **Property** table from within the InstallShield interface. Windows Installer properties give you access to many machine-specific variables such as the user's name or company.

In Advanced UI and Suite/Advanced UI projects, the Property Manager view lets you set Advanced UI or Suite/Advanced UI properties that you want to be available to the Advanced UI or Suite/Advanced UI installation at run time.

InstallScript View



Project *• The InstallScript view is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The InstallScript view enables you to customize your setup script using the InstallScript language.

Files, Functions, Properties, and Methods Folders

The InstallScript explorer in the center pane of the InstallScript view contains the following folders:

Table 12-98 • Folders in the InstallScript View

Folder	Project Type	Description
Files	Basic MSI, Merge Module, InstallScript MSI, InstallScript, InstallScript Object, Suite/Advanced UI	The Files folder lists all of your script (.rul) files. Click a script file to display its contents in the script editor. Your setup script must be named **Setup.rul**. You can include files with different names, but they must be included in **Setup.rul** with the #include preprocessor statement.
Functions	Basic MSI, Merge Module, InstallScript MSI, InstallScript, InstallScript Object, Suite/Advanced UI	The Functions folder contains a list of the InstallScript functions in all of the script files of your project. Click a function to display that function in the script editor.  ***Note*** • *If auto completion for local variables is disabled in the script editor, the Functions folder does not list any functions from your script files. To learn how to enable auto completion for local variables, see Enabling or Disabling Auto Completion in the Script Editors.*
Properties	InstallScript, InstallScript Object	The Properties folder contains a list of the InstallScript properties in all of the script files of your project. Click a property to display that property's declaration in the script editor. (The property's procedures are listed in the Functions folder.)  ***Note*** • *If auto completion for local variables is disabled in the script editor, the Properties folder does not list any properties from your script files. To learn how to enable auto completion for local variables, see Enabling or Disabling Auto Completion in the Script Editors.*
Methods	InstallScript Object	The Methods folder contains a list of the InstallScript methods in all of the script files of your project. Click a method to display that method in the script editor.  ***Note*** • *If auto completion for local variables is disabled in the script editor, the Methods folder does not list any methods from your script files. To learn how to enable auto completion for local variables, see Enabling or Disabling Auto Completion in the Script Editors.*

Script Editor

Clicking an item under any of the folders in the InstallScript view displays the script editor in the right pane in the InstallShield interface. To learn more, see Script Editor in the InstallScript View.

Script Editor in the InstallScript View



Project *• The script editor is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The script editor displays the text of the selected InstallScript file. You can modify or edit the script using commands from the Edit and Tools menus.

The Script Toolbar in the Script Editor



Project *• The script toolbar is available in the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

Basic MSI and Merge Module projects use custom actions instead of event handlers to run InstallScript code; Suite/Advanced UI projects use actions instead of event handlers to run InstallScript code.

The script toolbar is displayed across the top of the script editor in the InstallScript view; it contains an event category list box, as well as an event list box. The script toolbar lets you paste installation event handler function blocks in your script files. Event handlers in your script files override the default actions that are associated with installation events. You can modify the event handler code to change the actions performed during the installation.

Event Categories (list box, on the left)

Select the category containing the installation event whose event handler function block you want to paste in a script file. Events in the selected category are displayed in the Events list box.

Events (list box, on the right)

Select the installation event whose event handler function block you want to paste in a script file. If you select a feature event, its event handler is pasted in **FeatureEvents.rul**. If this file does not exist in your project, selecting a feature event automatically creates the file. If you select some other type of installation event, its event handler is pasted in **Setup.rul**.

If you change the default feature event handler code in **FeatureEvents.rul**, you must put the following statement in **Setup.rul** to include your changes in the installation:

```
#include "FeatureEvents.rul"
```

Custom Actions and Sequences View (or Custom Actions View)



Project *• The Custom Actions and Sequences view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

This view is called the Custom Actions view in the following project types:

- *DIM*
- *Merge Module*
- *MSM Database*

The Custom Actions and Sequences view has three separate areas: a Custom Actions area, an Action Text area, and a Sequences area. The Custom Actions view has the Custom Actions area.

Custom Actions

Microsoft enables you to add flexibility to your installation that is not directly supported by Windows Installer. This additional functionality is achieved through the use of custom actions.

InstallShield supports the use of custom actions to run InstallScript, VBScript, or JavaScript code; call DLL functions; run executable files; call a managed method in a managed assembly; set a property or a directory; trigger an error and end the installation; run PowerShell scripts; terminate a process; or run other installation packages.

The Custom Actions explorer in the Custom Actions and Sequences view behaves the same way as the Custom Actions explorer in the Custom Actions view. For descriptions of each type of custom action, see Custom Action Types. For a description of each custom action setting, see Custom Action Settings.

Action Text

To keep end users informed, installations commonly display text on the progress dialog to describe the installation's current activity. This usually accompanies the progress bar as a means of installation status. As each standard action and custom action is encountered, a message about the action is displayed on the progress dialog. This may be especially useful for actions that take a long time to execute. The same action text is also written to a log file if one is created at run time.

The Action Text explorer in the Custom Actions and Sequences view (for Basic MSI, InstallScript MSI, MSI Database, and Transform projects) lets you specify the action text for any action in your project. For a description of each action text setting, see Action Text Settings.



Important *• The names of the action text items under the Action Text explorer should match the names of standard and custom actions that are in your project. If you change the name of a custom action, you must also change the name of its action text item; otherwise, the action's text is not displayed at run time or written to the installation's log file.*

Sequences

Sequences direct all of the actions that are performed during the installation process—from file transfer to user interface display (for Basic MSI, MSI Database, and Transform projects). These actions are given a number in the sequence, which then executes from the lowest number to the highest. Rather than having to manually provide a numeric value for every action, you can use the Custom Actions and Sequences view to insert actions into a sequence, or edit the sequence timeline.



Project *• In InstallScript MSI projects, the installation's user interface is generated through the script; the dialogs are not inserted into sequences in the Custom Actions and Sequences view.*

If you are creating a custom action or custom dialog in a DIM project or a Merge Module project, you need to first import that DIM or module into an installation project and then add it to a sequence through the Custom Actions and Sequences view of that installation project.

There are three main sequences into which you can insert your dialogs (in Basic MSI, MSI Database, and Transform projects) and custom actions.

- Installation Sequence
- Advertisement Sequence
- Administration Sequence

Each of these sequences plays a different role. The Installation sequence is run during a normal installation. The Advertisement sequence runs when an application is being advertised rather than installed. The Administration sequence is run during an administrative installation.

Modifying Sequences, Actions, and Dialogs

The Custom Actions and Sequences view supports drag-and-drop editing and copying. The context menus in this view provide additional editing methods.

- To sequence a new custom action, drag it from the Custom Actions explorer to the appropriate location in a sequence under the Sequences explorer. When you drop it, drop it onto the item that should be directly before it in the sequence.
- To move a dialog, standard action, or custom action to a different point in a sequence (or from one sequence to another), drag it from the old location and drop it onto the item that should be directly before it in the sequence.
- To copy a custom action from one sequence to another, press and hold CTRL while dragging the custom action from one sequence to another sequence, and drop it onto the action or dialog that should be directly before it.
- When you select a dialog or standard action in the Sequences explorer, the Sequence tab on the right enables you to change the sequence number and add or modify the associated conditions.
- When you select a custom action in the Sequences explorer, more than one tab is displayed on the right:
  - Sequence tab—Use this tab to change the sequence number and to add or modify any conditions for the action.
  - Action tab—Use this tab to change any settings for the custom action.
  - Script tab—This tab is displayed for VBScript and JScript custom actions. It has a script editor that lets you edit your VBScript or JScript code.

- To edit the layout or behavior of a dialog, right-click the dialog in the Sequences explorer and then click Edit Behavior or Edit Layout.

Note • *A custom action cannot be called twice in the same sequence, since the custom action name is the key in the* ***CustomAction*** *table. Therefore, you cannot move or copy a custom action to a sequence that already contains that custom action.*

Dialogs and standard actions cannot be moved to a different type of sequence through a drag-and-drop operation.

Custom Action Types

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

Note that some custom action types are not available in some project types. Project-specific differences are noted where appropriate.

InstallShield includes support for several kinds of custom actions:

Table 12-99 • Custom Actions Supported in InstallShield

Icon	Action Type	Project Type	Custom Action Behavior
	**InstallScript**	Basic MSI, InstallScript MSI, Merge Module	Run InstallScript code.
	**EXE**	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Launch an executable file.
	**Standard DLL**	Basic MSI, InstallScript MSI	Call a function in a standard DLL.

Table 12-99 • Custom Actions Supported in InstallShield (cont.)

Icon	Action Type	Project Type	Custom Action Behavior
	**MSI DLL**	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Call a function in a DLL that was written specifically for Windows Installer. The entry point of the .dll file must have a predefined parameter and return value.
	**Managed Code**	Basic MSI, DIM, InstallScript MSI, Merge Module	Call a public method in a managed assembly that is written in managed code such as Visual Basic .NET or C#. For more information, see Calling a Public Method in a Managed Assembly.
	**Set Property**	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Set a property in the **Property** table. This is useful if you need to get information from the end user and store it so that it can be used later in your installation.
	**Set Directory**	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Set a directory in the **Directory** table at run time. For example, if you want to create a temp directory that is a subdirectory of the installation directory selected by the end user, you can use this option to set that new temp directory for use later in the installation.
	**Nested MSI**	Basic MSI, InstallScript MSI, MSI Database, Transform	Launch another .msi package. This is also known as a nested installation.  ***Important*** • *Nested installations is a deprecated feature of the Windows Installer. Applications installed with nested installations sometimes fail because they are difficult for end users to service correctly. Microsoft Corporation recommends that you avoid using nested installations and nested-installation custom actions to install products that are intended to be released to the public. To learn more, see Concurrent Installations in the Windows Installer Help Library.*
	**VBScript**	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Run 32-bit or 64-bit VBScript code.

Table 12-99 • Custom Actions Supported in InstallShield (cont.)

Icon	Action Type	Project Type	Custom Action Behavior
	**JScript**	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Run 32-bit or 64-bit JScript code.
	**Error**	Basic MSI, DIM, InstallScript MSI, Merge Module, MSI Database, MSM Database, Transform	Display a specified error message, return failure, and end the installation. For example, you may want to add an error custom action to your project to handle scenarios where an end user tries to install the current version of your product over a future major version. For more information, see Preventing the Current Installation from Overwriting a Future Major Version of the Same Product.
	**Kill Process**	Basic MSI, InstallScript MSI	Terminate a process at run time. For more information, see Calling a Kill-Process Custom Action.
	**PowerShell**	Basic MSI, InstallScript MSI	Run a PowerShell script to perform system configuration tasks at installation run time. For target system requirements, as well as other information about this type of custom action, see Calling a PowerShell Custom Action.
	**WiseScript**	Basic MSI	 ***Edition*** • *This custom action type is available in the version of InstallShield that is included in certain editions of AdminStudio.* Runs a WiseScript executable file. This type of action includes a wrapper MSI DLL that gets and sets Windows Installer properties and evaluates Windows Installer conditions.

To help you differentiate between immediate, deferred, commit, and rollback custom actions in your project, InstallShield adds color-coded dots to some of the action's icons. The dots are displayed on the icons for actions that are listed under the Custom Actions explorer in the Custom Actions and Sequences view or the Custom Actions view. The following table shows the icon for the executable file custom action.

Table 12-100 • Determining the In-Script Execution of a Custom Action by Viewing Its Icon

Icon	Description
	If a custom action does not have a dot, it indicates that the In-Script Execution setting is not applicable to that type of custom action, or one of the following values is selected: • Immediate Execution • Immediate Execution (Terminal Service Aware)
	If a custom action has a blue dot, one of the following values is selected for the action's In-Script Execution setting: • Deferred Execution • Deferred Execution (Terminal Service Aware) • Deferred Execution in System Context
	If a custom action has a green dot, one of the following values is selected for the action's In-Script Execution setting: • Commit Execution • Commit Execution (Terminal Service Aware) • Commit Execution in System Context
	If a custom action has a red dot, one of the following values is selected for the action's In-Script Execution setting: • Rollback Execution • Rollback Execution (Terminal Service Aware) • Rollback Execution in System Context

For details about each of the InstallShield custom actions that are added automatically to InstallShield projects to support different functionality, see InstallShield Custom Action Reference.



Windows Logo • *If you are applying for the Windows logo, any custom actions in your installation must follow best practices guidelines for custom action creation. You can use the InstallShield validation suites and the Full MSI Validation Suite to verify whether your installation package meets most of the custom action–related logo requirements. However, some of the requirements cannot be verified through the validation suite. To learn more, see Guidelines for Creating Custom Actions that Meet Requirements of the Windows Logo Program.*

Custom Action Settings

Project • *Use the Custom Actions and Sequences view to configure custom action settings in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Use the Custom Actions view to configure custom action settings in the following project types:

- *DIM*
- *Merge Module*
- *MSM Database*

Note that some custom action types are not available in some project types.

The settings in the Custom Actions and Sequences view (or Custom Actions view) are organized into the following categories:

- Action area settings
- Sequence area settings

To learn about displaying action information on the progress dialog and in the installation's log file, see Using Action Text.

Action Area Settings

The following custom action settings are available in the Action area.

Table 12-101 • Custom Action Settings in the Action Area

Setting	Description
Function Name	***Note*** • *This setting applies to InstallScript custom actions.* In the list of functions that are included in the project's InstallScript files, select the InstallScript function that you want to call.

Table 12-101 • Custom Action Settings in the Action Area (cont.)

Setting	Description
DLL File Name	 ***Note*** *• This setting applies to standard DLL custom actions that are stored in the* ***Binary*** *table or located in the target system's search path. This setting also applies to MSI DLL custom actions that are stored in the* ***Binary*** *table.* For the **Binary** table location: Enter the path to the DLL file that you would like to use for your custom action. As an alternative, you can select it from the list of available DLL files in the **Binary** table, or click the ellipsis button (...) to browse to it. For the target system's search path location: Enter the name of the DLL file that you would like to use for your custom action. As an alternative, you can click the ellipsis button (...) to browse to it. Only the file name is needed, since the custom action searches in the target system's search path to find this DLL. If the custom action type is an MSI DLL, the DLL file must comply with the required Windows Installer function signature. To learn more, see Calling Functions in Windows Installer DLL Files.
DLL Filekey	 ***Note*** *• This setting applies to standard and MSI DLL custom actions that are installed with the product.* In the list of DLL files that are included in your project, select the DLL file that you would like to use for your custom action.
Function Signature	 ***Note*** *• This setting applies to standard DLL custom actions.* To specify the parameters for the entry-point function in your DLL file, click the ellipsis button (...) in this setting. This opens the Function Signature dialog box, which has various settings that let you specify the function name, arguments, return type, and return property for the entry point function in your standard DLL.
Function Name	 ***Note*** *• This setting applies to MSI DLL custom actions.* Enter the name of the function in the MSI DLL that you would like to call.
Executable File Name	 ***Note*** *• This setting applies to executable file custom actions that are stored in the* ***Binary*** *table.* Enter the path to the executable file that you would like to use for your custom action. As an alternative, you can select it from the list of available executable files in the **Binary** table, or click the ellipsis button (...) to browse to it.

Table 12-101 • Custom Action Settings in the Action Area (cont.)

Setting	Description
Executable Filekey	 ***Note*** • *This setting applies to executable file custom actions that are installed with the product.* In the list of executable files that are included in your project, select the executable file that you would like to use for your custom action.
Working Directory	 ***Note*** • *This setting applies to executable file custom actions that are located in a path that references a directory.* Enter the working directory name for the executable file that you would like to use for your custom action, or select it from the list of available directories.
File Name & Command Line	 ***Note*** • *This setting applies to executable file custom actions that are located in a path that references a directory.* Enter path and name of the executable file that you would like to use for your custom action. If appropriate, you can include any command-line parameters that should be passed to the executable file. Use quotation marks around long file names. For example: • "[INSTALLDIR]subdirectory1\filename.exe" • "[WindowsFolder]Notepad.exe" • "[SUPPORTDIR]\fileFromSupportFilesView.exe" Note that the path is optional only if the executable file is already present in the operating system search path (that is, **WindowsFolder** or **SystemFolder**).
Executable Property	 ***Note*** • *This setting applies to executable file custom actions that are located in a path that is identified by a property.* Enter the name of the property that identifies the full path to your executable file, or select the property from the list of properties in your project.
Command Line	 ***Note*** • *This setting applies to executable file custom actions that are stored in the **Binary** table, installed with the product, or located in a path that is identified by a property.* Enter any command-line parameters that you would like to pass to your executable file. Enclose long file names within quotation marks.

Table 12-101 • Custom Action Settings in the Action Area (cont.)

Setting	Description
Assembly File	 ***Note*** *• This setting applies to custom actions that use a managed assembly whose location is set to* ***Binary*** *table.* Specify the .NET assembly file that you want to use for your managed-code custom action. You can select the **Browse to file** option to specify the file, or select the file from the list of DLL and .exe files that will be stored in the **Binary** file. At build time, InstallShield adds the managed assembly to a C++ Windows Installer wrapper DLL and adds the wrapper DLL to the **Binary** table of your .msi package. The wrapper DLL contains the information that is required to mediate, load, and run your assembly.
Assembly Filekey	 ***Note*** *• This setting applies to custom actions that use a managed assembly installed with the product.* To specify the managed assembly that you want to use for your custom action, select the file from the list of DLLs and .exe files that are included in your project.
Assembly Property	 ***Note*** *• This setting applies to custom actions that use a managed assembly whose path references a property or a directory in the* ***Directory*** *table.* To specify the managed assembly that you want to use for your custom action, do one of the following: • Select a Windows Installer property in the list. • Type the name of a new property. • Type the name of a directory in the **Directory** table. The property or directory name must be enclosed within square brackets ([]). You can add a string after the property if appropriate. The resulting path should indicate the location of the assembly file.
Method Signature	 ***Note*** *• This setting applies to custom actions that use a managed assembly.* Click the ellipsis button (...) to launch the Method Signature dialog box, which is where you specify arguments and return values for the managed method. When you have specified signature information on this dialog box, InstallShield uses it as the value of the Method Signature setting. To learn more, see Specifying the Signature for a Managed Method in an Assembly Custom Action.

Table 12-101 • Custom Action Settings in the Action Area (cont.)

Setting	Description
Error Message	 ***Note*** • *This setting applies to custom actions that trigger an error message.* Type the error message text that should be displayed when the custom action's conditions are met on the target system. As an alternative, you can type a property name whose value contains the error text. The property name must be enclosed within square brackets ([]).
Function Name	 ***Note*** • *This setting applies to kill-process custom actions.* Select the appropriate function: • **KillProcess**—If you selected one of the immediate options in the In-Script Execution setting and you want to kill a process that has a particular name, select this option. • **KillProcessByID**—If you selected one of the immediate options in the In-Script Execution setting and you want to kill a process that has a particular process identifier (PID), select this option. • **KillProcessDeferred**—If you selected one of the deferred, commit, or rollback options in the In-Script Execution setting and you want to kill a process that has a particular name, select this option. • **KillProcessByIDDeferred**—If you selected one of the deferred, commit, or rollback options in the In-Script Execution setting and you want to kill a process that has a particular PID, select this option. For information on configuring this type of custom action, see Calling a Kill-Process Custom Action.
PowerShell File Name	 ***Note*** • *This setting applies to PowerShell custom actions that are stored in the* ***Binary*** *table.* Select the PowerShell script file (.ps1) in the list of files that are stored in the **Binary** table or that are included in your project. If the location that you specified is stored in the **Binary** table, you can click this ellipsis button (...) in this setting to browse to the file. For target system requirements, as well as other information about this type of custom action, see Calling a PowerShell Custom Action.

Table 12-101 • Custom Action Settings in the Action Area (cont.)

Setting	Description
PowerShell Script FileKey	 ***Note*** • *This setting applies to PowerShell custom actions that use a PowerShell script that is installed with the product.* To specify the PowerShell script file (.ps1) that you want to use for your custom action, select the file from the list of PowerShell files that are included in your project. For target system requirements, as well as other information about this type of custom action, see Calling a PowerShell Custom Action.
WiseScript Executable	 ***Note*** • *This setting applies to WiseScript custom actions.*  ***Edition*** • *This custom action type is available in the version of InstallShield that is included in certain editions of AdminStudio.* Enter the path to the WiseScript executable file that you would like to use for your custom action. As an alternative, you can click the ellipsis button (...) to browse to it.
Processes	 ***Note*** • *This setting applies to kill-process custom actions.* Enter executable file names or PIDs of the processes that you want to terminate before proceeding with the installation. Separate multiple processes by semicolons. • If **Function Name** is **KillProcess** or **KillProcessDeferrred**, then specify executable file names in the **Processes** setting. • If **Function Name** is **KillProcessByID** or **KillProcessByIDDeferred**, then specify PIDs in the **Processes** setting. For information on configuring this type of custom action, see Calling a Kill-Process Custom Action.

Table 12-101 • Custom Action Settings in the Action Area (cont.)

Setting	Description
Return Processing	Specify how the custom action thread should be processed. Valid options are: • Asynchronous (No wait for completion) • Asynchronous (Wait for exit code) • Synchronous (Check exit code) • Synchronous (Ignore exit code) These flags are used to specify that the main and custom action threads run synchronously (the installer waits for the custom action thread to complete before resuming the main installation thread) or asynchronously (the installer runs the custom action simultaneously as the main installation continues). Note that some options are not applicable to some types of custom actions. Only options that pertain to the selected type of custom action are available in this list.
In-Script Execution	Select which iteration of the sequence will trigger your action. For details about each option, see In-Script Execution. These options copy the action code into the execution, rollback, or commit script. If you select a Terminal Server Aware option, the custom action impersonates the user during per-machine installations on terminal server machines.

Table 12-101 • Custom Action Settings in the Action Area (cont.)

Setting	Description
Execution Scheduling	Specify how many times you want the action to run. For example, if an action is listed in both the user interface and execute sequences, it can be set to run both times, or it can run only once. Available options are: • **Always execute**—The action runs every time it is encountered. Therefore, it could run in the user interface sequence and again in the execute sequence. • **Execute only if running on client after UI sequence**—The action runs only if the execute sequence is run after the user interface sequence. • **Execute only once**—The action runs only once—even if it exists in both the user interface and execute sequences. The action is skipped in the execute sequence if the user interface sequence has run. • **Execute only once per process**—The action runs once per process if it exists in both the user interface and execute sequences. If the execute sequence runs in the same process as the user interface sequence, the installation skips this action in the execute sequence. This scheduling lets you avoid rerunning actions that modify session state such as properties. If you selected anything other than Immediate execution in the In-Script Execution setting, the Execution Scheduling setting is not available and is set to Always execute.  ***Project*** • *The* ***Execute only once*** *option does not work for InstallScript MSI projects. If your custom action exists in both the user interface and execute sequences, it is executed twice. This is because—in an InstallScript MSI project—the InstallScript engine executes the user interface sequence and Windows Installer executes the execute sequence.*
Silent Mode	 ***Note*** • *This setting applies to standard DLL custom actions.* Specify whether you want the installation to suppress a warning message if the custom action fails to load the DLL file and call the function.
MSI Type Number	This read-only setting identifies the Windows Installer numeric value for the selected custom action. For more information about basic custom action types, see Custom Action Type Overview.
Comments	Enter any internal comments that you want to record for this custom action. These comments are for your reference only and are not displayed to end users.

Table 12-101 • Custom Action Settings in the Action Area (cont.)

Setting	Description
Help File Path	Click the ellipsis button (...) to browse to the file that describes the behavior of the custom action. The file should be a text-based file such as a .txt, .htm, or .rtf file. At build time for Basic MSI, DIM, InstallScript MSI, and Merge Module projects, InstallShield streams the contents of this file into the Description column of the **ISCustomActionReference** table. If you are working on a project in Direct Edit mode, InstallShield streams the contents of the file that you select into the Description column of the **ISCustomActionReference** table as soon as you select the help file. In addition, InstallShield makes this setting read-only and displays `[Text Data]` as the value for this setting.  ***Windows Logo*** • *The intended behavior of each custom action must be documented for the Windows logo program. This is especially helpful if system administrators deploy your product to enterprise environments; they sometimes need to know what the custom actions do. If you validate your installation package but you have not specified a value for the Help File Path setting, InstallShield generates validation error ISICE10. For more information, see ISICE10.* For any built-in InstallShield custom actions, InstallShield makes this setting read-only and displays `InstallShield Custom Action` as the value.
Run During Patch Uninstall	 ***Project*** • *If you are working on a project in Direct Edit mode, this setting is not applicable unless the database schema is a minimum of 405 (for Windows Installer 4.5 or later).* Specify whether Windows Installer should run the custom action only when a patch is being uninstalled. You can select Yes for a custom action in an .msi package. You can also select Yes for a new custom action that is added by a patch. However, Yes should not be selected for a custom action that is being added or removed by a patch to an existing custom action. The default value for this setting is No. When Windows Installer 4.5 runs the patch uninstall custom action, it uses the custom action in the patch that is being uninstalled.  ***Note*** • *Windows Installer 4.5 includes support for this setting, but Windows Installer 4.0 and earlier do not. Therefore, if you select Yes for this setting and some target systems may have Windows Installer 4.0 or earlier, add a condition to this custom action to prevent Windows Installer 4.0 and earlier from running it. Otherwise, Windows Installer 4.0 and earlier would call the custom action during the installation, repair, or update of the package.* For the condition, use the **MSIPATCHREMOVE** property. For more information, see MSIPATCHREMOVE in the Windows Installer Help Library.

Sequence Area Settings

The following custom action settings are available in the Sequence area.

Table 12-102 • Custom Action Settings in the Sequence Area

Setting	Description
Install UI Sequence	To schedule this custom action during the user interface sequence of the installation mode, select the appropriate option. To leave this custom action out of the sequence, select **<Absent from sequence>**. As an alternative, you can schedule an action by dragging it from the Custom Actions explorer to the appropriate location in a sequence under the Sequences explorer. When you drop it, drop it onto the item that should be directly before it in the sequence. To move it to a different point in a sequence (or from one sequence to another), drag it from the old location and drop it onto the item that should be directly before it in the sequence.
Install UI Condition	If you want to specify a condition that Windows Installer must evaluate before running the custom action during this sequence on target systems, enter the conditional statement in this setting. The custom action does not run if the condition evaluates to false. As an alternative to manually entering a statement, you can click the ellipsis button (...) in this setting to launch the Condition Builder dialog box, which simplifies the process of creating a condition. For more information, see Building Conditional Statements.
Install Exec Sequence	To schedule this custom action during the execute sequence of the installation mode, select the appropriate option. To leave this custom action out of the sequence, select **<Absent from sequence>**. As an alternative, you can schedule an action by dragging it from the Custom Actions explorer to the appropriate location in a sequence under the Sequences explorer. When you drop it, drop it onto the item that should be directly before it in the sequence. To move it to a different point in a sequence (or from one sequence to another), drag it from the old location and drop it onto the item that should be directly before it in the sequence.
Install Exec Condition	If you want to specify a condition that Windows Installer must evaluate before running the custom action during this sequence on target systems, enter the conditional statement in this setting. The custom action does not run if the condition evaluates to false. As an alternative to manually entering a statement, you can click the ellipsis button (...) in this setting to launch the Condition Builder dialog box, which simplifies the process of creating a condition. For more information, see Building Conditional Statements.

Table 12-102 • Custom Action Settings in the Sequence Area (cont.)

Setting	Description
Advertise Exec Sequence	To schedule this custom action during the execute sequence of the advertisement mode, select the appropriate option. To leave this custom action out of the sequence, select **<Absent from sequence>**. As an alternative, you can schedule an action by dragging it from the Custom Actions explorer to the appropriate location in a sequence under the Sequences explorer. When you drop it, drop it onto the item that should be directly before it in the sequence. To move it to a different point in a sequence (or from one sequence to another), drag it from the old location and drop it onto the item that should be directly before it in the sequence.
Advertise Exec Condition	If you want to specify a condition that Windows Installer must evaluate before running the custom action during this sequence on target systems, enter the conditional statement in this setting. The custom action does not run if the condition evaluates to false. As an alternative to manually entering a statement, you can click the ellipsis button (...) in this setting to launch the Condition Builder dialog box, which simplifies the process of creating a condition. For more information, see Building Conditional Statements.
Admin UI Sequence	To schedule this custom action during the user interface sequence of the administration mode, select the appropriate option. To leave this custom action out of the sequence, select **<Absent from sequence>**. As an alternative, you can schedule an action by dragging it from the Custom Actions explorer to the appropriate location in a sequence under the Sequences explorer. When you drop it, drop it onto the item that should be directly before it in the sequence. To move it to a different point in a sequence (or from one sequence to another), drag it from the old location and drop it onto the item that should be directly before it in the sequence.
Admin UI Condition	If you want to specify a condition that Windows Installer must evaluate before running the custom action during this sequence on target systems, enter the conditional statement in this setting. The custom action does not run if the condition evaluates to false. As an alternative to manually entering a statement, you can click the ellipsis button (...) in this setting to launch the Condition Builder dialog box, which simplifies the process of creating a condition. For more information, see Building Conditional Statements.

Table 12-102 • Custom Action Settings in the Sequence Area (cont.)

Setting	Description
Admin Exec Sequence	To schedule this custom action during the execute sequence of the administration mode, select the appropriate option. To leave this custom action out of the sequence, select **<Absent from sequence>**. As an alternative, you can schedule an action by dragging it from the Custom Actions explorer to the appropriate location in a sequence under the Sequences explorer. When you drop it, drop it onto the item that should be directly before it in the sequence. To move it to a different point in a sequence (or from one sequence to another), drag it from the old location and drop it onto the item that should be directly before it in the sequence.
Admin Exec Condition	If you want to specify a condition that Windows Installer must evaluate before running the custom action during this sequence on target systems, enter the conditional statement in this setting. The custom action does not run if the condition evaluates to false. As an alternative to manually entering a statement, you can click the ellipsis button (...) in this setting to launch the Condition Builder dialog box, which simplifies the process of creating a condition. For more information, see Building Conditional Statements.

For more information about these settings, see CustomAction Table in the Windows Installer Help Library.

Action Text Settings



Project • *Use the Custom Actions and Sequences view to configure action text settings in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

To keep end users informed, installations commonly display text on the progress dialog to describe the installation's current activity. This usually accompanies the progress bar as a means of installation status. As each standard action and custom action is encountered, a message about the action is displayed on the progress dialog. This may be especially useful for actions that take a long time to execute. The same action text is also written to the installation's log file if one is created.

The Action Text explorer in the Custom Actions and Sequences view lets you specify the action text for any action in your project.

When you click a custom action item or a standard action item in the Action Text area of the Custom Actions and Sequences view, the following settings are available.

Table 12-103 • Action Text Settings

Setting	Description
Description	Enter text that describes the selected action. For example, the default description for the InstallFiles action is: `Copying new files` When the Windows Installer launches this action, the text that you enter is displayed on the progress dialog. For more information, see Displaying Action Descriptions on the Progress Dialog. If a log file is created at run time, this description is written to the log file when the Windows Installer launches this action. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 12-103 • Action Text Settings (cont.)

Setting	Description
Template	Enter the template that should be used to format action data records. For example, the default template for the InstallFiles action is: `File: [1], Directory: [9], Size: [6]` When the Windows Installer launches this action, the action details may be displayed on the progress dialog. For the aforementioned InstallFiles example, the [1] property is replaced with the names of files as they are transferred to the target system. The [9] property is replaced with the directory that contains the file, and the [6] property is replaced with the file's size in bytes. Note that by default, the action data is not displayed on the progress dialog. For more information, see Displaying Action Data on the Progress Dialog. If a template is not specified for the selected action, no data is displayed when the action is launched. In addition, the installation must be run with a full user interface; if the installation is run silently or with a reduced or basic user interface, the action data is not shown. To find out what data fields are available for use in the Template setting value of a particular standard action, see Standard Actions Reference in the Windows Installer Help Library, and then refer to the help for the specific action.  ***Project*** • *InstallScript MSI installations cannot display action data on the status dialog. However, the template is written to the installation's log file.*  ***Note*** • *The value of this setting must always be* **[1]** *for several standard actions:* • *GenerateScript* • *Rollback* • *RollbackCleanup* *Therefore, the Template setting is disabled for these actions.* When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Support Files View (Advanced UI, Basic MSI, InstallScript Object, and Suite/Advanced UI Projects)



Project • *This information applies to the following project types:*

- *Advanced UI*

- *Basic MSI*
- *InstallScript Object*
- *Suite/Advanced UI*

The Support Files view is where you can add support files to your project, and—in Basic MSI projects—add files to disk image folders.

The Support Files view contains the following main areas:

Support Files

The Support Files area lets you add, sort, and delete support files—files that are required by your installation only during the installation process. The installation removes the support files when the installation is complete. Add any language-dependent files to the appropriate language-specific area. Add language-independent files to the Language Independent area.

Splash Screen



Project *• The Splash Screen area is available for Basic MSI projects.*

The Splash Screen area lets you add to your project a startup graphic—the graphic that is displayed when the end user begins the installation. The graphic file must be a bitmap file (.bmp). Add the file to the appropriate language-specific or language-independent folder.



Note *• In a single-language installation, the splash screen that is specified for the single language (or the language-independent splash screen if no splash screen is specified for the single language) is displayed.*

In a multilanguage installation, the splash screen for the language in which the installation is running is displayed. If no splash screen is specified for the language in which the installation is running, the language-independent splash screen is used.

Only one splash screen is displayed to the end user during the installation. If you have more than one splash screen file in the language-independent area or for a specific language, the first file in the list is displayed at run time.

Advanced Files



Project *• The Advanced Files area is available for Basic MSI projects.*

The Disk1 item in the Advanced Files area enables you to indicate files and folders that you want to go on Disk1 of your installation media. These files and folders are not automatically installed to the target system when your installation is run. Rather, you can link to the installation media from your application or from the installation. For example, you might include a large redistributable file with your application that you do not want included in the application installation. This file should be placed in the Disk1 folder.

Support Files/Billboards View (InstallScript and InstallScript MSI Projects)

Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The Support Files/Billboards view is where you can add support files and billboard files to your project, and add files to disk image folders.

The Support Files/Billboards view contains the following main areas:

Support Files

The Support Files area lets you add, sort, and delete support files—files that are required by your installation project only during the installation process. Add any language-dependent files and folders to the appropriate language-specific area. Add language-independent files and folders to the Language Independent area.

In your InstallScript code, support files are uncompressed into a temporary directory and then deleted when the installation is complete.

Billboards

The Billboards item enables you to specify files that you want to display to the end user during the installation process. The billboards can be used to communicate, advertise, educate, and entertain end users. To learn more, see Displaying Billboards in InstallScript and InstallScript MSI Installations.

Splash Screen

Project *• In InstallScript projects, you can use .bmp and .gif files as splash screen files. In InstallScript MSI projects, only .bmp files can be used as splash screen files.*

The Splash Screen area lets you add to your project a startup graphic—the graphic that is displayed when the end user begins the installation. Add the file to the appropriate language-specific or language-independent folder.

Note *• In a single-language installation, the splash screen that is specified for the single language (or the language-independent splash screen if no splash screen is specified for the single language) is displayed.*

In a multilanguage installation, the splash screen for the language in which the installation is running is displayed. If no splash screen is specified for the language in which the installation is running, the language-independent splash screen is used.

Only one splash screen is displayed to the end user during the installation. If you have more than one splash screen file in the language-independent area or for a specific language, the first file in the list is displayed at run time.

Project *• In an InstallScript project, the splash screen file must be named* **Setup.bmp** *or* **Setup.gif***. This file name requirement does not apply to InstallScript MSI projects.*

Advanced Files

The Advanced Files item enables you to indicate files and folders that you want to go on a disk of your installation media. These files and folders are not automatically installed to the target system when your installation program is run. Rather, you can link to the installation media from your application or from the installation. For example, you might include a large redistributable file with your application that you do not want included in the application installation.

Table 12-104 • Advanced File Items

Item	Description
Disk1	Add files and folders that you want to go on the first disk of your installation media.
Last Disk	Add files and folders that you want to go on the last disk of your installation media.  ***Project*** • *This applies to InstallScript projects.*
Other	Add files and folders that you want to go on a specific disk of your installation media; you specify the disk at build time in the General Options panel of the Release Wizard.  ***Project*** • *This applies to InstallScript projects.*

System Search View



Project • *The System Search view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The System Search view provides the Windows Installer capability to search for a particular file, folder, registry key, .ini file value, or .xml file value on a target system prior to installation. This functionality lets you perform application, version, and configuration data searches.

The System Search view displays a grid listing each search that you want to conduct on the target system. You can use this view to add a predefined system search—whether it is a search that is included with InstallShield or one that is stored in a repository—to your project. You can also use the System Search view to customize any predefined searches or define your own system searches for your project.

Whenever you define your own search, the System Search Wizard is launched. From there, you can select from a list of search options and specify search details, such as the number of subfolder levels to search. When you modify an existing search, you can alter your initial selections in the System Search Wizard.

Property Manager View



***Project** • The Property Manager view is available in the following Windows Installer–based projects:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

This view is also available in Advanced UI and Suite/Advanced UI projects.

Note that some differences between the Property Manager view in Windows Installer–based projects and that in Advanced UI and Suite/Advanced UI projects exist; these differences are noted where appropriate.

The Windows Installer engine and the Advanced UI and Suite/Advanced UI engine maintain global information using properties. The Property Manager view displays a list of installation properties that are used by the Windows Installer engine or the Advanced UI and Suite/Advanced UI engine at run time. You can modify, create, and delete installer properties in the Property Manager view. At build time, InstallShield writes the properties in the Property Manager view to installation that it creates.

Following is a list of some of the tasks that you can perform in this view:

- View the properties that are defined in your project.
- Add, modify, and delete properties.
- Filter the properties that are shown in the view to hide ones that do not contain a specific string.
- Resize and reorder the columns in the view.
- Sort the rows in the view by any column by clicking the column heading.
- Drag and drop column headings on to the group box area (the area below the view's buttons) to organize the rows in the view in a hierarchical format.
- Windows Installer–based projects: Make a property localizable so that it can have different values based on the language that your installation uses.



***Project** • In Windows Installer–based projects, properties that appear in all uppercase letters are called public properties, and they can be changed by the end user on the command line at run time. All others must be set before the release is built or—at run time—through a custom action or dialog's behavior.*

Working with the Property Manager View

The Property Manager view consists of the following elements:

- A row of buttons and other controls
- A group box area (below the row of buttons)
- A spreadsheetlike table

Each row in the table represents a property in your project.

The following table describes all of the buttons and other controls that are displayed in the Property Manager view.

Table 12-105 • Controls in the Property Manager View

Name of Control	Icon	Description
New Property		Adds a new property to your project.  ***Project*** • *In Windows Installer–based projects: To create a localizable property, you can click the arrow next to this button, and then click Localizable Property. To learn more, see Creating a Localizable Property.*
Delete Selected Properties		Deletes the selected row or rows.
Clear Selected Properties		Deletes the value of the selected properties.
Make Selected Properties Localizable		 ***Project*** • *This button is available in Windows Installer–based projects.* Adds a string identifier to the value of the selected properties, enabling you to set a different property value for every language that your project supports. To learn more, see Making an Existing Property Localizable.
Expand All Groups		Shows all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Collapse All Groups		Hides all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Search Grid		Dynamically filters the properties that are displayed in the Property Manager view according to the string that you specify in this search box. As you type a string in this box, InstallShield hides all of the rows that do not contain it.

Table 12-105 • Controls in the Property Manager View (cont.)

Name of Control	Icon	Description
Property Manager Help		Displays the help for the Property Manager view.
Drag a column header here to group that column		Use this group box area to group rows in the view. The view supports multiple levels of grouping simply by dragging the column headings and dropping them onto the group box. InstallShield displays the rows in the view hierarchically according to column arrangement in the group box. To learn more, see Working with the Group Box Area in Various Views.

The following table describes each of the columns in the Property Manager view.

Table 12-106 • Columns in the Property Manager View

Column	Description
Name	This column shows the name of the Windows Installer or Advanced UI and Suite/ Advanced UI property.
Value	This column shows the value of the property.  ***Project*** • *In Windows Installer–based projects: If the property is configured to be localizable, this column shows the string identifier in curly brackets before the string value.*
Comments	This column contains an internal note about the properties. The comments are not displayed at run time.

Table 12-106 • Columns in the Property Manager View (cont.)

Column	Description
Formatted	 ***Project*** *• This column is available in the following project types:* • *Advanced UI* • *Suite/Advanced UI* This column contains a check box that lets you indicate whether the value in the Value column is a formatted expression that contains property names, environment variable references, or other special strings that you want to be resolved at run time. The run-time resolution occurs early at run time, before the installation executes the actions in the OnBegin event. To leave the property value as is at run time, clear this check box. Square brackets and other characters have no special meaning. To replace the formatted expressions at run time, select this check box. To learn about the syntax that is available for formatted expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

Events View



Project *• The Events view is available in Suite/Advanced UI projects.*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Events view enables you to create your own actions and add them to your Suite/Advanced UI project. You can also use this view to schedule an action to occur during one or more run-time events, or you can use the Packages view to schedule an action to occur immediately before or after a specific package in your installation is run.



Tip *• Optionally, you can associate actions with wizard interface elements of the Suite/Advanced UI installation. To learn more, see Configuring an Action for an Element in the Wizard Interface.*

The Events view has two separate area: an Actions tab and an Events area.

Actions

Use the Actions explorer in the Events view to define and configure actions in your project. You can create actions that run executable files, call DLL functions, run PowerShell scripts, set a Suite/Advanced UI property, run InstallScript code, or call a public method in a managed assembly.

For a description of each action setting, see Events View Settings.

Events

Events direct all of the actions that are performed during install mode, maintenance mode, and stage-only mode. The Suite/Advance UI engine executes actions that are scheduled for an event in the order that they are listed in this view. For descriptions of each type of event, see Types of Events in a Suite/Advanced UI Installation.

Events View Settings

Project • *The Events view is available in Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The settings in the Events view are organized into the following categories:

- Action area settings on the Action tab
- Return Process Handling area settings on the Action tab for an .exe action
- Events area settings

If you select a PowerShell action in the Actions explorer or the Events explorer, InstallShield displays a PowerShell Script tab on the right; you can use this tab to edit your script.

Action Area Settings on the Action Tab

The following action settings are available in the Actions area on the Action tab.

Table 12-107 • Action Settings in the Action Area on the Action Tab

Setting	Description
File	***Note*** • *This setting applies to .exe, DLL, and managed-code actions.* Enter the run-time path to the file that you want to use for your action. As an alternative, if the file is not present on target systems at run time, and if none of the packages in your Suite/Advanced UI installation are going to install it, you can click the ellipsis button (...) in this setting to browse to it. If you browse to the file, InstallShield adds it to your project as a support file in the Support Files view of your Suite/Advanced UI project. At run time, the file is available on the target system only during your product's installation process.

Table 12-107 • Action Settings in the Action Area on the Action Tab (cont.)

Setting	Description
Source Path	 ***Note*** • *This setting applies to PowerShell actions.* Enter the source path to the PowerShell script (.ps1) that you want to use for your action. As an alternative, you can click the ellipsis button (...) in this setting to browse to it. When you browse to the file, InstallShield adds it to your project as a support file in the Support Files view of your Suite/Advanced UI project. At run time, the file is available on the target system only during your product's installation process. Instead of hard-coding a location, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.  ***Tip*** • *Use the PowerShell Script tab in this view to edit the code for this action.*
Arguments	 ***Note*** • *This setting applies to .exe actions.* If appropriate, specify the command-line parameters that should be used with the action. Separate multiple parameters with spaces.
Function Name	 ***Note*** • *This setting applies to DLL and InstallScript actions.* For a DLL action: Enter the name of the function in the DLL that you would like to call. For an InstallScript action: In the list of functions that are included in the project's InstallScript files, select the InstallScript function that you want to call.
Class	 ***Note*** • *This setting applies to managed-code actions.* Enter the name of the class for the managed method that you want the selected action to call. To learn more, see Working with a Managed-Code Action in a Suite/Advanced UI Installation.

Table 12-107 • Action Settings in the Action Area on the Action Tab (cont.)

Setting	Description
Method	 ***Note*** • *This setting applies to managed-code actions.* Enter the public method that you want the selected action to call. To learn more, see Working with a Managed-Code Action in a Suite/Advanced UI Installation.
Requires Administrative Privileges	 ***Note*** • *This setting applies to .exe, DLL, PowerShell, InstallScript, and managed-code actions.* Specify whether this action requires administrative privileges on systems that have Windows Vista or later or Windows Server 2008 or later. To learn more, see Minimizing the Number of User Account Control Prompts During Advanced UI and Suite/Advanced UI Installations.
Verb	 ***Note*** • *This setting applies to .exe actions.* Specify the verb that should be used with the selected action's file, or select the appropriate verb from the list. If you leave this setting blank, the verb that is the default one on the target system for this file is used. If the default verb is not available, the *open* verb is used. If neither verb is available, the first verb that is listed in the registry is used. The use of the runas verb is not recommended. If the action requires administrative privileges, select Yes for the action's Requires Administrative Privileges setting. For more information, see Minimizing the Number of User Account Control Prompts During Advanced UI and Suite/Advanced UI Installations.

Table 12-107 • Action Settings in the Action Area on the Action Tab (cont.)

Setting	Description
Window	 ***Note*** • *This setting applies to .exe actions.* Select the initial window state of the .exe action when it is launched. Available options are: • **Show**—Display the window in its current size and position. • **Hide**—Hide the window. • **Minimize**—Display the window as minimized. • **Maximize**—Display the window as maximized. • **Normal**—Display the window in its original size and position.
Wait for Exit	 ***Note*** • *This setting applies to .exe actions.* Specify whether you want the installation to wait for the action to complete before proceeding with the next part of the installation. Available options are: • **Yes**—The installation waits for the action to complete before resuming. This is the default option. • **No**—The installation runs the action simultaneously as the next part of the installation proceeds. If you select Yes for this setting, you can use the settings in the Return Process Handling area to specify the various codes that the action may return.
Property Name	 ***Note*** • *This setting applies to actions that set a property.* Enter the property name, or select it from the list of available properties. The property names that are displayed in the list are the ones in the Property Manager view. You can enter any other property name and it will be available at run time. If you want to explicitly create a new property, you can do so from the Property Manager view.

Table 12-107 • Action Settings in the Action Area on the Action Tab (cont.)

Setting	Description
Property Value	 ***Note*** • *This setting applies to actions that set a property.* Enter the value of the property. If you select Yes for the Format Property Value setting, the value that you enter can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
Format Property Value	 ***Note*** • *This setting applies to actions that set a property.* Specify whether the value that you enter in the Value setting is a formatted expression that contains property names, environment variable references, or other special strings that you want to be resolved at run time. Available options are: • **Yes**—Replace the formatted expressions at run time. • **No**—Leave the property value as is at run time. The value that you are entering does not include formatted expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.

Return Process Handling Area Settings on the Action Tab for an .exe Action

The following action settings are available in the Return Process Handling area on the Action tab for an .exe action.

Table 12-108 • Return Process Handling Settings in the Action Area on the Action Tab for an .exe Action

Setting	Description
Ignore Code	 ***Note*** • *This setting applies to .exe actions.* Specify a comma-delimited list of the codes that the action could return when the installation should proceed with the next part of the installation. This setting is available if you select Yes in the action's Wait for Exit setting.

Table 12-108 • Return Process Handling Settings in the Action Area on the Action Tab for an .exe Action (cont.)

Setting	Description
Abort Code	 ***Note*** • *This setting applies to .exe actions.* Specify a comma-delimited list of the codes that the action could return when the installation should be aborted. Note that if the action cannot be launched or loaded, the installation is aborted. This setting is available if you select Yes in the action's Wait for Exit setting.
Reboot Code	 ***Note*** • *This setting applies to .exe actions.* Specify a comma-delimited list of the codes that the action could return when the target system should be restarted. This setting is available if you select Yes in the action's Wait for Exit setting.
Cancel Code	 ***Note*** • *This setting applies to .exe actions.* Specify a comma-delimited list of the codes that the action could return when the installation should be cancelled. This setting is available if you select Yes in the action's Wait for Exit setting.
Default Response	 ***Note*** • *This setting applies to .exe actions.* Select the option that describes how you want the installation to respond if the code that the action returns is not specified for any of the other settings in the Return Process Handling area. Available options are: • **Ignore**—The installation proceeds with the next part of the installation. This is the default option. • **Abort**—The installation is aborted. • **Reboot**—The target system is restarted. • **Cancel**—The installation is cancelled. This setting is available if you select Yes in the action's Wait for Exit setting.

Event Area Settings

The following action settings are available in the Events area.

Table 12-109 • Action Settings in the Events Area

Setting	Description
Condition	This setting lets you specify one or more conditions that the Suite/Advanced UI installation should use to evaluate whether the action should be run. For example, if the action should be run only on certain platforms, you can create a platform requirement in a condition for this action; the Suite/Advanced UI would launch the action only on appropriate platforms. To add one or more new action conditions, click the New Condition button in this setting. InstallShield adds a new row under the Condition setting. Select the appropriate option—All, Any, or None—from the list in this row. Then in this row, click the New Condition button, and select the appropriate option to continue building the conditional statement. ***Tip*** • *If you define a Feature Operation condition or a Package Operation condition for an action, the earliest event for which you can schedule the action is OnStaging.* For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects. If one or more conditional statements are configured, the Condition setting says **(Condition)**. If none are configured, the Condition setting says **(Empty)**.
Display Text	***Note*** • *This setting applies to .exe, DLL, PowerShell, InstallScript, and managed-code actions.* Enter text that describes the selected action. For example, if the action is running a script that configures the target system, you could enter the following string: `Configuring the system` If the installation launches the action when the InstallationProgress wizard page is displayed, the text that you enter is displayed on the wizard page. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

User Interface View



Project *• The User Interface view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Suite/Advanced UI*
- *Transform*

The appearance of your installation is one of the main aspects that differentiates your product from that of your competition. You can easily customize the way your installation looks through the views listed below.

Dialogs



Project *• The Dialogs view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

In the Dialogs view, you can select dialogs to display at run time. The way in which you control a dialog's behavior depends on which engine controls the user interface: the Windows Installer engine displays the dialogs for Basic MSI projects, and the InstallScript engine displays the dialogs for InstallScript and InstallScript MSI projects. For more information, see:

- Dialogs View (InstallScript, InstallScript MSI, and InstallScript Object Projects)
- Dialogs View (Basic MSI and Other Windows Installer–Based Projects)

Wizard Interface



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Use the Wizard Interface view to customize the user interface of your Advanced UI or Suite/Advanced UI installation. This view lets you add wizard pages to and remove wizard pages from the default list of pages, as well as customize the layout and behavior of each page.

Billboards

Project *• The Billboards view is available in Basic MSI projects.*

For information about billboard support in InstallScript or InstallScript MSI projects, see Support Files/Billboards View (InstallScript and InstallScript MSI Projects).

Billboards are images or Adobe Flash application files that are displayed for a specified amount of time during the file transfer portion of your installation. The billboards can be used to communicate, advertise, educate, and entertain end users. For example, billboards can present overviews on new features of the product being installed or other products from your company. Each billboard is a file that you or your company's graphics department creates for complete control over the look and feel of the file transfer.

String Editor

Project *• The String Editor view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

To help streamline the process of localizing a project, all of the text strings that may be displayed at run time during the installation process are available in one consolidated view: the String Editor view. You can use this view to edit the strings for everything from button text to feature descriptions. You can also use this view to export each language's string entries to a file, translate the values that are listed in the file, and then import the translated file into your project.

Dialogs View (InstallScript, InstallScript MSI, and InstallScript Object Projects)



Project *• This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The Dialogs view contains a list of the standard end-user dialogs. The dialogs are identified by their function names. Click a dialog in this view to view a sample dialog in the right pane.

The dialog names in the list are *ghosted* until you edit the layout. A ghosted name means that the default dialog (from **isres.dll**) will be used in the user interface. If you edit the layout of a dialog, its name appears in bold in the list, and the dialog is pulled from **isuser.dll** at run time.

If your installation project supports additional languages, those languages appear as nodes beneath the edited dialog. You can edit the layout for each language separately.



Tip *• Selecting and editing a dialog in the Dialog Editor does not automatically place the dialog in the end-user interface. In order for a dialog to be displayed at run time, it needs to be included in the InstallScript code. For example, code for the dialogs that are displayed during a first-time installation are part of the* ***OnFirstUIBefore*** *and* ***OnFirstUIAfter*** *event handlers. To learn more, see Displaying Dialogs During InstallScript and InstallScript MSI Installations.*

Dialog Resource Files

By default, all dialogs in InstallScript and InstallScript MSI projects are kept in a resource .dll file called **_IsRes.dll**. This .dll file is built into your distribution package when you build the installation. There is a separate **_IsRes.dll** for every supported language; they are located in the InstallShield folder's Redist folder (for example, *`InstallShield Program Files Folder`*\Redist). None of the **_IsRes.dll** files should ever be modified.

When you edit a dialog in the Dialog Editor, InstallShield makes a copy of the dialog from the original **_IsRes.dll** file. This copy is stored in your project file. When your project is built, all edited dialogs (and any new dialogs you have created) are built into a resource .dll file called **_IsUser{CurrentLanguage}.dll**. For English, a .dll file called **_IsUser1033.dll**. **_IsUser{LanguageID}.dll** is then built into your distribution package.

At run time, the engine that displays the dialogs first looks for a dialog in **_IsUser{LangId}.dll**. If the dialog is not found, it then looks for the default version in **_IsRes.dll**. The engine looks for dialogs based on their resource identifier. You can see the resource identifier of a dialog by going to the **Dialog** table in the Direct Editor.

Dialogs View (Basic MSI and Other Windows Installer–Based Projects)



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *Merge Module*
- *MSI Database*
- *MSM Database*
- *Transform*

The Dialogs view contains a Dialogs explorer. The Dialogs explorer contains two main folders: Themes and All Dialogs.

Themes

The Themes folder contains a list of all of the dialog themes that you can use for the dialogs in your project. A dialog theme is a predefined set of images that give your end-user dialogs a unified and distinctive look.

All Dialogs

The All Dialogs folder contains a list of the dialogs in your project. Each dialog item in the Dialogs explorer contains a Behavior item, which enables you to configure the behavior that is associated with a dialog, and at least one language item, which displays the dialog in the Dialog Editor of the selected language. The Dialog Editor enables you to edit the layout for each language separately.

When you create a new Basic MSI project, InstallShield provides a series of default dialogs for the two User Interface sequences in which a Windows Installer package typically displays dialogs: the Installation sequence (which runs when the installation is launched in the default mode, for example, by double-clicking an .msi file)—with separate dialogs depending on whether the package is being installed for the first time, reinstalled, or uninstalled—and the Administration sequence (the list of actions that are executed when you launch the installation with the `/a` command-line option). (End-user dialogs are not usually displayed in the Advertisement sequence, which contains the list of actions that are executed when you launch the installation with the `/j` command-line option.)



***Project** • Although you can create dialogs in DIM and Merge Module projects, you cannot add them to a sequence until you add the DIM or merge module to an installation project. Then, all the dialogs included in your module are available to be included.*

Wizard Interface View



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Wizard Interface view contains a Wizard Interface explorer. The Wizard Interface explorer contains the following main areas: Styles, Brushes, Control Themes, Wizard Pages, and Secondary Windows.

Styles

The Styles area of the Wizard Interface view lets you configure sets of formatting characteristics that you can apply to one or more parts of the wizard interface to quickly change its appearance. The Styles area lets you define the following types of styles:

- **Font sets**—A font set is a collection of fonts (including attributes such as font name, size, and weight). For each font in a font set, you can specify to which language the font is applicable. This enables you to select a different font for each language that your project supports.
- **Text styles**—A text style defines text attributes such as text color, as well as font name, style, and size. Text styles are typically applied to the text that is displayed in the Advanced UI or Suite/Advanced UI wizard interface.

Font sets work in conjunction with text styles. Text styles reference a font set but can optionally override various font attributes that are defined at the font set level.

To add a style to your project, right-click the Styles node and then click the appropriate command, depending on the type of style that you want to add.

To learn more, see the following:

- Defining a Custom Style for the Wizard Interface
- Applying a Text Style to Text in the Wizard Interface

Brushes

A brush defines a solid color, a gradient, or an image for various elements of the wizard interface, such as the background of wizard pages and controls.

To add a brush to your project, right-click the Brushes node and then click Add Brush.

To learn more, see the following:

- Defining a Custom Brush for the Wizard Interface
- Specifying the Background Brush for the Header, Body, and Navigation Areas of the Wizard Interface
- Overriding the Default Background Brushes for a Wizard Page or Secondary Window

Control Themes

A control theme is a collection of colors for various parts of controls—text color, background color, and border color—and for various states of controls—such as clicked and hovered. You can specify the control theme that you want to use for all of the controls in your user interface by default. You can also override the theme for specific controls on wizard pages and windows in your user interface. Control themes give buttons and other items in your installation's user interface a flat look (without three-dimensional effects) that is reminiscent of the style of Windows Store apps.

To add a control theme to your project, right-click the Control Themes node and then click Add Control Theme.

To learn more, see the following:

- Defining a Custom Control Theme for the Wizard Interface
- Applying a Theme to a Control in the Wizard Interface

Wizard Pages

The main type of window that is displayed to end users of an Advanced UI or Suite/Advanced UI installation is a wizard page. The Wizard Pages area of the Wizard Interface view lets you view, edit, add, and remove wizard pages in your Advanced UI or Suite/Advanced UI project.

To configure and edit settings that apply to all of the wizard pages in your Advanced UI or Suite/Advanced UI project, click the Wizard Pages node. The settings that are displayed for this node let you specify dimensions, the caption that is used in the title bar of the wizard pages, and more.

The specific wizard page subnodes under the Wizard Pages node let you preview, sequence, and remove pages as needed; these subnodes also let you modify the layout of each wizard page—adding, moving, and removing a variety of different kinds of controls. To change the run-time order of the wizard pages, click a page that you want to resequence, and then press CTRL+UP ARROW or CTRL+DOWN ARROW until the page reaches the appropriate place under the Wizard Pages node.

To add a wizard page to your project, right-click the Wizard Pages node and then click the appropriate command, depending on the type of page that you want to add.

Secondary Windows

A secondary window in an Advanced UI or Suite/Advanced UI installation is a pop-up window that lets end users perform a command, asks end users a question, or provides end users with information. One example of a secondary window is a FilesInUse window, which informs end users about one or more applications that are open and locking files that need to be updated by the installation. The specific window subnodes under the Secondary Windows node let you preview and remove pages as needed; these subnodes also let you modify the layout of each window, just as you can edit the layout of wizard pages.

To add a secondary window to your project, right-click the Secondary Windows node and then click Add Blank Window.

Settings for the Wizard Interface

For information about various settings for each area in the Wizard Interface view, see the help panes that are displayed when you click any setting in this view.

Wizard Interface View Toolbar



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

If you select a wizard page or secondary window in the Wizard Interface view, the toolbar that InstallShield shows directly above the wizard interface preview pane includes several different buttons and other controls that let you modify the layout of the selected page or window. InstallShield also displays this toolbar in the Wizard Interface view if you select one or more controls on a wizard page or a secondary window.

Table 12-110 • Wizard Interface View Toolbar

Name of Control	Icons	Description
New Control		Lets you add a control to the selected wizard page or secondary window. Select one of the control buttons, or the control types that are listed in the drop-down list next to the control buttons.
New Control (cont.)		The control types that are available from the drop-down list next to the Label button are: • **Label**—The Label control displays text. Use this type of control for labels that you want to position next to other controls—such as a text box control—and for instructions or information that you want to include on a wizard page or secondary window. • **Hyperlink**—The hyperlink control displays an HTML link; clicking the link at run time opens a page in the default browser on the target system. • **Group Box**—A group box can be used to enclose various controls in one area. A group box also provides a label that you can use to express the relationship between the controls that are contained within it. • **Frame**—A frame control lets you draw the outline of a box. • **Image**—An image control displays an image. • **Billboard**—A billboard control displays one or more images that can be updated in response to a property such as the **ISInstallProgress** property, or configured to display for a predetermined duration. You can add billboards to your projects to display information to end users during the installation process. The billboards can be used to communicate, advertise, educate, and entertain end users. • **Progress Bar**—A progress bar is a dynamic bar that fills up in response to a property such as the **ISInstallProgress** property. • **Progress Circle**—A progress circle is a pair of dynamic circles that fill up in response to two properties: a property such as **ISParcelProgress**, which indicates the completion percentage of the current package, and a property such as **ISInstallProgress**, which indicates the completion percentage of the overall entire installation. The outside progress circle typically shows the overall progress, and the inside progress bar typically shows the progress of the entire package.

Table 12-110 • Wizard Interface View Toolbar (cont.)

Name of Control	Icons	Description
New Control (cont.)		The control types that are available from the drop-down list next to the Text Box button are: • **Text Box**—A text box control is a field in which end users can enter text and numbers. • **Password Box**—A password box control is a field in which end users can enter a password. • **Rich Text Box**—A rich text box is a read-only control that displays text from an .rtf file. This type of control is often used on license agreement or readme wizard pages.
New Control (cont.)		The control types that are available from the drop-down list next to the Button button are: • **Button**—A button control is a control that carries out a command when an end user clicks it. • **Command Link**—A command link control offers a set of mutually exclusive options, similar to radio buttons. Each option includes an icon, a command link label next to the icon, and an optional supplemental explanation about the option. • **Check Box**—A check box control displays a check box that an end user can select or clear to toggle on or off an option. • **Radio Button**—A radio button control is one of two or more mutually exclusive options that end users can select. A radio button must be inserted into a group box control; it functions as part of that control. • **Image Button**—An image button control displays an image, and it carries out a command when an end user clicks it.

Table 12-110 • Wizard Interface View Toolbar (cont.)

Name of Control	Icons	Description
New Control (cont.)		The control types that are available from the drop-down list next to the Combo Box button are: • **Combo Box**—A combo box control is a box that contains a drop-down list of predefined values. The box is also a text box that lets end users enter a custom value. To change this control to a drop-down list without the text box (that is, if end users should be able to select a predefined value but not enter a custom value), set the CBS_DROPDOWNLIST style for this control to True. • **List Box**—A list box control is a standard list box that lets end users select a single option from a list of predetermined options. • **Checked List Box**—A checked list box is a list box in which each of the predefined values is associated with a check box. A checked list box lets end users select one or more options, or leave all of the options unselected. • **Feature Selection Tree**—A feature selection tree control is a special tree view control that contains a hierarchically organized collection of the Advanced UI or Suite/Advanced UI installation's features; it lets end users select one or more features to install.
Position Controls		Lets you change the position of two or more selected controls.
Position Controls (cont.)		The position change options that are available from the drop-down list next to the Align Left button are: • **Align Left**—Align the left edges of two or more selected controls with the left edge of the control that was selected last. • **Align Center**—Align the center of two or more selected controls with the center of the control that was selected last. • **Align Right**—Align the right edges of two or more selected controls with the right edge of the control that was selected last. • **Align Top**—Align the top edges of two or more selected controls with the top edge of the control that was selected last. • **Align Middle**—Align the middle of two or more selected controls with the middle of the control that was selected last. • **Align Bottom**—Align the two or more selected controls with the bottom of the control that was selected last. • **Distribute Horizontally**—Position three or more selected controls so that they are an equal distance from each other horizontally. • **Distribute Vertically**—Position three or more selected controls so that they are an equal distance from each other vertically.

Table 12-110 • Wizard Interface View Toolbar (cont.)

Name of Control	Icons	Description
Position Controls (cont.)		The position change options that are available from the drop-down list next to the Make Same Width button are: • **Make Same Width**—Increase or decrease the width of each of the selected controls so that each width matches that of the control that was selected last. • **Make Same Height**—Increase or decrease the height of each of the selected controls so that each height matches that of the control that was selected last. • **Make Same Size**—Increase or decrease the dimensions of each of the selected controls so that each control's dimensions match that of the control that was selected last.
Default Language		Lets you switch the strings that InstallShield displays on the wizard pages and secondary windows in this view to those in a different language in your project. The available language options are the ones that are selected in the Setup Languages setting in the General Information view. This setting also lets you change the default language for the project.

Billboards View



Project • *The Billboards view is available in Basic MSI projects.*

For information about billboard support in InstallScript or InstallScript MSI projects, see Support Files/Billboards View (InstallScript and InstallScript MSI Projects).

You can add billboards to your projects to display information to end users during the installation process. The billboards can be used to communicate, advertise, educate, and entertain end users. For example, billboards can present overviews on new features of the product being installed or other products from your company. Each billboard is a file that you or your company's graphics department creates for complete control over the look and feel of the file transfer.

If you add one or more billboards to your project, the billboards are displayed at run time as the SetupProgress dialog reports the modifications that Windows Installer is making to the system. You can control the length of time the billboard is shown and its position by configuring its settings in the Billboards view.

To learn about the settings in the Billboards view, see:

- Billboard Settings—These are project-wide billboard settings.
- Settings for Adobe Flash Application File Billboards and Image Billboards—These settings are displayed in the right pane in the Billboards view when you click a Flash billboard or image billboard in the center pane.

Billboard Settings



Project • *The Billboards view is available in Basic MSI projects.*

When you click the Billboards explorer in the center pane of the Billboards view, InstallShield displays the following settings in the right pane. These are project-wide settings for billboards.

Table 12-111 • Billboard Settings

Setting	Description
Billboard Type	Select the type of billboard that you want to use for your installation. Available options are: • **Fullscreen with Small progress (displayed in lower right)**—When the installation displays the standard end-user dialogs, it also displays a full-screen background. During file transfer, the installation shows full-screen backgrounds, with billboards in the foreground, and a small progress box in the lower-right corner of the screen. • **Windowed with Standard progress**—During file transfer, the installation displays a standard-size dialog that shows the billboards. The bottom of this dialog shows the progress bar. The installation does not display a background for this style. • **Windowed with Small (displayed in lower right, no billboards)**—The installation displays a small progress box in the lower-right corner of the screen during file transfer, but it does not display any billboards or a background. For more information, including sample screen shots of each billboard type, see Types of Billboards for Basic MSI Projects.
Loop Billboards	Specify whether you want your installation to continuously loop the image billboards until the file transfer completes and the installation shows the appropriate SetupComplete dialog. If you select No for this setting and the file transfer takes more time than you have allocated for the billboards, the installation continues displaying the last image billboard until the file transfer ends. If you select Yes for this setting and the file transfer takes more time than you have allocated for the billboards, the installation restarts the display of billboards from the beginning. The loop continues, if necessary, until the file transfer ends. The default value for this setting is No. This setting has no effect on Adobe Flash application file billboards.

Settings for Adobe Flash Application File Billboards and Image Billboards



Project • *The Billboards view is available in Basic MSI projects.*

A Flash or image billboard's settings determine which file is shown, how long it is displayed, and its position on the screen. To access these settings, open the Billboards view, and in the Billboards explorer, select the billboard that you want to configure.

Table 12-112 • Settings for Adobe Flash Application File Billboards and Image Billboards

Setting	Description
File Name	Do one of the following: • **For an Adobe Flash application file billboard**—Enter the path to the Flash application file (.swf) that you would like to use for the selected billboard, or click the ellipsis button (...) to browse to the file. Flash application files can consist of videos, movies, sounds, interactive interfaces, games, text, and more—anything that is supported by the .swf type of file. It is recommended that files such as Flash video files (.flv) and MP3 audio files be embedded in the .swf file so that they are available locally on the target system during file transfer. Although .swf files can reference external files that you can post on a Web site, this external implementation would require that end users have an Internet connection. • **For an image billboard**—Enter the path to the image file (.bmp, .gif, .jpg, or .jpeg) that you would like to use for the selected billboard, or click the ellipsis button (...) to browse to the file. Note that animated .gif files are not supported. If you want to use animation in a billboard, consider using an Adobe Flash application file billboard.  ***Note*** • *If the version of Flash or other tool that you use to create your .swf file is newer than the version of the Flash Player that is installed on a target system, it is possible that some of the Flash features may not work as expected on that target system.*
Duration	Enter the amount of time, in seconds, that this billboard should be displayed. The number that you enter must be from 1 to 32767 (which is a little more than 9 hours). The effect that the duration has on the run-time behavior depends on whether the installation is displaying a Flash billboard or image billboards. To learn more, see Run-Time Behavior of a Basic MSI Installation that Includes Billboards.

Table 12-112 • Settings for Adobe Flash Application File Billboards and Image Billboards (cont.)

Setting	Description
Origin	Select where on the screen you want your billboard to be anchored. Available options are: • Upper Right • Upper Left • Lower Right • Lower Left • Centered The X and Y coordinates are measured from this point.  ***Note*** • *This setting is used only if the* ***Fullscreen with Small progress (displayed in lower right)*** *option is selected for the Billboard Type setting.*
X Coordinate	To change the horizontal location of your billboard relative to the location you selected for the Origin setting, enter the distance, in pixels. For example, if the billboard's origin is Lower Left, an X Coordinate value of 100 places the left side of the billboard 100 pixels from the left side of the screen.  ***Note*** • *This setting is used only if the* ***Fullscreen with Small progress (displayed in lower right)*** *option is selected for the Billboard Type setting.*
Y Coordinate	To change the vertical location of your billboard relative to the location you selected for the Origin setting, enter the distance, in pixels. For example, if the billboard's origin is Lower Left, a Y Coordinate value of 100 places the bottom of the billboard 100 pixels from the bottom of the screen.  ***Note*** • *This setting is used only if the* ***Fullscreen with Small progress (displayed in lower right)*** *option is selected for the Billboard Type setting.*
Effect	Select the transition effect for this billboard. Rather than just appearing on the screen and disappearing after an allotted amount of time, a transition effect makes the change between billboards much smoother.  ***Note*** • *This setting is applies to image billboards, but not to Adobe Flash application file billboards.* *In addition, this setting is used only if the* ***Windowed with Standard Progress*** *option or the* ***Fullscreen with Small progress (displayed in lower right)*** *option is selected for the Billboard Type setting.*

Table 12-112 • Settings for Adobe Flash Application File Billboards and Image Billboards (cont.)

Setting	Description
Background Color	This setting displays the currently selected background color for your billboard. To change this color, click the ellipsis button (...). InstallShield displays the Color dialog box, which lets you select a predefined color or define a custom color for the background. ***Note*** *• This setting is used only if the* ***Fullscreen with Small progress (displayed in lower right)*** *option is selected for the Billboard Type setting.*
Title	Enter the title of this billboard, as you want it to appear in the upper-left corner of the background. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. ***Tip*** *• The maximum number of characters that the installation can display for the title at run time varies, depending on the font, the font size, the font attributes, and the length of the title string that you specify for this setting. It also depends on the screen resolution on the target system. Therefore, if you specify a long title, preview the billboard using different screen resolutions to test whether the entire title will be displayed at run time.* ***Note*** *• This setting is used only if the* ***Fullscreen with Small progress (displayed in lower right)*** *option is selected for the Billboard Type setting.*
Background Style	Select the style of background that you would like to use. Available options are: • **Gradient**—The background fades from dark to light. • **Solid**—The background is displayed as one solid color. ***Note*** *• This setting is used only if the* ***Fullscreen with Small progress (displayed in lower right)*** *option is selected for the Billboard Type setting.*
Font	Click the ellipsis (...) button to select the font that you want to use for the title in the selected billboard's background. If the target machine does not have the font you selected, a default system font is used instead. ***Note*** *• This setting is used only if the* ***Fullscreen with Small progress (displayed in lower right)*** *option is selected for the Billboard Type setting.*

String Editor View



Project *• The String Editor view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

In the String Editor view, you have complete and centralized control over the localizable text strings that are displayed at run time during the installation process. You can use this view to edit the strings for everything from button text to feature descriptions.

The String Editor view shows the collection of language-independent identifiers and corresponding language-specific values for your project. Following is a list of some of the tasks that you can perform in this view:

- View all of the strings in your project.
- Add, modify, and delete strings.
- Filter the strings that are shown to hide ones that do not contain a specific string.
- Resize and reorder the columns in the view.
- Sort the rows in the view by any column by clicking the column heading.
- Drag and drop column headings on to the group box area (the area below the view's buttons) to organize the rows in the view in a hierarchical format.
- Export all of the string identifiers and their corresponding values into a text file (.txt).
- Import the translated strings from a .txt file into the project.
- Search the project to identify all of the instances in which a specific string identifier is used. Find out if a string is not used anywhere within a project.

Working with the String Editor View

The String Editor view consists of the following elements:

- A row of buttons and other controls
- A group box area (below the row of buttons)
- A spreadsheetlike table

Each row in the table represents a string entry in your project.

The following table describes all of the buttons and other controls that are displayed in the String Editor view.

Table 12-113 • Controls in the String Editor View

Name of Control	Icon	Description
New String Entry		Displays the String Entry dialog box, which lets you add a new string entry.
Edit Selected String		Displays the String Entry dialog box, which lets you edit the selected string entry.
Delete Selected Strings		Deletes the selected row or rows.
Expand All Groups		Shows all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Collapse All Groups		Hides all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Find String		Displays the Find dialog box, which lets you search for instances of a string. This dialog box lets you specify criteria such as whether you want to match the case.
Find Next		Searches for the next occurrence of the specified string.
Find and Replace		Displays the Replace dialog box, which lets you search for instances of a string and replace them with a new string. This dialog box lets you specify criteria such as whether you want to find or ignore a string with specific capitalization.
Search for Selected Strings in Project		Searches the entire project for all instances of the string whose row is selected, and shows the search results in the Output window.
Export Strings		Lets you export all of the strings for a particular language to a text file (.txt). You can provide that .txt file to a translator who can update the file with translated text.
Import Strings		Lets you select the text file (.txt) that contains the strings that you want to import into your project. Also lets you specify the language for those strings.
Search Grid		Dynamically filters the strings that are displayed in the String Editor view according to the string that you specify in this search box. As you type a string in this box, InstallShield hides all of the rows that do not contain it.

Table 12-113 • Controls in the String Editor View (cont.)

Name of Control	Icon	Description
Default Language		Lets you view and change the default language for the project.
String Editor Help		Displays the help for the String Editor view.
Drag a column header here to group that column		Use this group box area to group rows in the view. The view supports multiple levels of grouping simply by dragging the column headings and dropping them onto the group box. InstallShield displays the rows in the view hierarchically according to column arrangement in the group box. To learn more, see Working with the Group Box Area in Various Views.

The following table describes each of the columns in the String Editor view.

Table 12-114 • Columns in the String Editor View

Column	Description
Language	This column shows the language of the string entry.
Identifier	This column contains the language-independent ID for the string. Each string identifier in a project is linked to one or more values.
Value	This column shows the run-time string.  ***Project*** • *In Basic MSI, InstallScript MSI, and Merge Module projects, some of the string values contain Windows Installer properties inside square brackets—for example,* ***Install [ProductName]****. At run time, the property and brackets are replaced by the property value.* *String values in these same project types may also contain font information in curly brackets—for example,* ***{&MSSansBold8}OK****. The font information indicates style details that should be used to display the strings at run time.*
Comments	This column contains an internal note about the string entries. The comments are not displayed at run time.
Modified	This column lists the date and time that the string entry was last modified.

Media View



Project • *The Media view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The final step in creating your installation project is to build and test your installation. InstallShield provides you with many different media types to choose from, as well as the ability to test your installation from within InstallShield.

Path Variables



Project *• The Path Variables view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

With path variables, you can define commonly used paths in a central location so that you do not need to change every source file's path each time you move the project or change the directory structure.

Upgrades



Project *• The Upgrades view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Upgrades view provides a visual, integrated method for adding and authoring settings within the **Upgrade** table of your .msi database.

Releases



Project *• The Releases view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The Releases view enables you to build different configurations of your installation, test the user interface, or launch your installation for a trial run.

Patch Design



***Project** • The Patch Design view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*

In the Patch Design view, you can create a Windows Installer patch package (.psp) to update your application on an end user's system.

Path Variables View



***Project** • The Path Variables view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *QuickPatch*
- *Suite/Advanced UI*

The traditional way to link to source files in an installation project is to create a reference to that file using a hard-coded path. For example, you might have a source file called **Program.exe** located at **C:\Work\Files** that you want to include in your installation.

Using Hard-Coded Paths

If you use hard-coded paths, you have to enter the entire path every time you want to associate a source file from that directory. If you move the file to another directory, you have to change the hard-coded path as it appears in your installation project. If your installation consisted of a small number of source files, this might not be a problem. Unfortunately, some installations contain thousands of files that all need to be remapped if you change the folder structure or migrate the project to a different machine.

Using Path Variables

With path variables, you can define commonly used paths in a central location so that you do not need to change every source file's path each time you move the project or change the directory structure. In the previous example, if you keep all of your application's source files in various subfolders under **C:\Work\Files**, you could create one variable that points to the Files folder—*<MyFiles>*. If you want to include a file that is in **C:\Work\Files\Images**, you enter **<MyFiles>\Images**. If you move your files to **D:\Work\Files**, you can go to one place, your variable *<MyFiles>*, and change the folder that it points to.

All path variables can be viewed and modified in the Path Variables view. You can use path variables in almost any location in InstallShield where you link to source files, such as in the Dialog Editor, dynamic file links, and the release location. Instead of entering the path variables yourself, you can have InstallShield recommend them whenever you browse to a path.



Note • *Path variables are used during the development of your installation project. These paths do not apply to the target machines where the application is being installed. Rather, they are used to link to source files for your installation project. When the project is built, those links are evaluated and the files they point to will be built into the installation.*

Path Variables Types

There are four types of path variables that you can use. Each type functions somewhat differently from the others. Regardless of the path variable type you use, the variable name is provided in the same manner throughout InstallShield.

Table 12-115 • Types of Path Variables

Variable Type	Description
	Predefined path variables are path variables that point to some of the most commonly used folders. Unlike other types of path variables, these values cannot be edited in InstallShield. For more information, see Predefined Path Variables.
	The values of registry-based path variables are derived from the registry keys you created. After creating the registry key, you need to set a path variable to this key. For more information, see Registry Path Variables.
	Environment path variables are based on the values of your system's environment variables. You can set an environment path variable to an existing environment variable. For more information, see Environment Path Variables.
	Standard, or user-defined, path variables are defined through InstallShield. You can specify a path variable such as <MyFiles> with a value of C:\Work\Files. These variables do not rely on any outside sources, such as the registry or system paths. For more information, see Standard Path Variables.

You also have the option of converting existing static links to path variables with the Convert Source Paths Wizard. This wizard scans your installation project for static links and changes those links to path variables, which makes your project more easily portable.

Working with the Path Variables View

The Path Variables view consists of the following elements:

- A row of buttons and other controls
- A group box area (below the row of buttons)
- A list of path variables that are defined in your project

The following table describes all of the buttons and other controls that are displayed in the Path Variables view.

Table 12-116 • Controls in the Path Variables View

Name of Control	Icon	Description
New Path Variable		Adds a new standard path variable to your project. To create a registry path variable or an environment variable, you can click the arrow next to this button, and then click the appropriate command. To learn more, see Creating and Defining a Custom Path Variable.
Delete Selected Path Variables		Deletes the selected variable or variables.
Refresh		Refreshes the list of variables in the view.
Expand All Groups		Shows all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Collapse All Groups		Hides all of the rows in the groups if you are using groups to organize the rows in a hierarchical format.
Show Group Box		Shows or hides the group box area below the row of buttons in this view.
Search Grid		Dynamically filters the path variables that are displayed in the Path Variables view according to the string that you specify in this search box. As you type a string in this box, InstallShield hides all of the rows that do not contain it.
Path Variables View Help		Displays the help for the Path Variables view.

Table 12-116 • Controls in the Path Variables View (cont.)

Name of Control	Icon	Description
Drag a column header here to group that column		Use this group box area to group rows in the view. The view supports multiple levels of grouping simply by dragging the column headings and dropping them onto the group box. InstallShield displays the rows in the view hierarchically according to column arrangement in the group box. To learn more, see Working with the Group Box Area in Various Views.

The following table describes each of the columns in the Path Variables view.

Table 12-117 • Columns in the Path Variables View

Column	Description
Name	In this column, enter the name of your variable. You do not need to enter angle brackets, but the path variable is displayed in angle brackets whenever it is used in a path. For example, if your variable is named *MyRegVar* and a component's files are linked to the folder contained in its value, the component's Link To folder shows **<MyRegVar>**.

Table 12-117 • Columns in the Path Variables View (cont.)

Column	Description

Defined Value	In this column, define the value of the path variable.
	
	For Standard Path Variables
	
	For standard variables, enter the directory to which you would like your variable to point.
	
	
	
	Note *• You can also refer to other path variables in the defined value by enclosing the referenced path variable name in angled brackets. For example, if you have a path variable called* `MyRoot` *with a value of* **C:**, *you can refer to it in a path variable definition for another variable, such as* `Games`*. The actual path for the* `Games` *variable might be* **C:\Programs\GameFiles,** *but you can define* `Games` *as* **<MyRoot>\Programs\GameFiles**. *However, if you attempt to self-reference a path variable, the literal string is used instead. For example, defining* `Games` *as* **<MyRoot>\Programs\<Games>** *actually results in* `Games` *defined as* **C:\Programs\<Games>**.
	
	For Registry Path Variables
	
	Enter the complete registry key, with the final "subkey" being the name of the value whose data contains the folder. For example, define *MyRegVar* as follows:
	
	`HKEY_LOCAL_MACHINE\SOFTWARE\TestKey\TestValue`
	
	Assume that TestKey has the following subkey and values:
	
	`[HKEY_LOCAL_MACHINE\SOFTWARE\TestKey]`
	`@="C:\\MyPath1"`
	`"TestValue"="C:\\MyPath2"`
	`[HKEY_LOCAL_MACHINE\Software\TestKey\TestValue]`
	`@="C:\\MyPath3"`
	
	Even though HKEY_LOCAL_MACHINE\SOFTWARE\TestKey has a subkey called TestValue, *MyRegVar* points to the value TestValue, and the current value will be **C:\MyPath2**. (Note, however, that if a value named TestValue does not exist, InstallShield reads the default value (**C:\MyPath3**) of the subkey HKEY_LOCAL_MACHINE\SOFTWARE\TestKey\TestValue.)
	
	For Environment Path Variables
	
	For environment path variables, enter the name of the variable as it appears on the Environment dialog box.

Current Value	This is a read-only column that contains the actual path that the variable points to. This is a useful reference for environment and registry path variables, since they are defined outside InstallShield.

Table 12-117 • Columns in the Path Variables View (cont.)

Column	Description
Test Value	 ***Project*** • *This column applies to the following project types:* • *Basic MSI* • *DIM* • *InstallScript MSI* • *Merge Module* This column enables you to enter a hard-coded path that can be used when the registry path variable is resolved. To use test values in your release, you must select the **Use path variable test values** option on the Advanced Settings panel of the Release Wizard. This option is deselected by default. When this option is selected, the test value is used for all registry path variables. Otherwise, the test value is not used at all.
Type	This column shows the type of variable (predefined, standard, environment, or registry). Click a field in this column to change from one type of path variable to another. Note that you cannot change the type of a predefined path variable.



Tip • *To override the value of a custom standard path variable, an environment path variable, or a registry path value for a particular release at build time, use the Path Variable Overrides setting on the Build tab for that release. To learn more, see Overriding the Value of a Custom Path Variable for a Release.*

Upgrades View



Project • *The Upgrades view is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Project-specific differences are noted where appropriate.

The Upgrades view provides a visual, integrated method for adding and authoring settings within the **Upgrade** table of an .msi database.

To add an upgrade item, right-click the Upgrade Windows Installer Setup node. The available options are:

Table 12-118 • Upgrades View Options

Option	Description
Major Upgrade Item	A major upgrade effectively uninstalls the existing installation of a product, and then installs the latest product version. A major upgrade is appropriate for substantial installation architecture changes that may or may not change the major version number of the product (such as upgrading version 1.1.0 to version 2.0.0). A major upgrade is required if any of the following are true: • An existing component's component code has changed, or a component has been removed from the product tree. • The key file of an existing component has changed. • An existing feature has been moved in the product tree, or deleted from the product tree. • The name of the .msi file has changed. For more information, see Creating Major Upgrades.
Minor Upgrade Item	Small updates and minor upgrades are functionally identical except the product version changes for a minor upgrade, but not a small update. A minor upgrade installs over an existing application while a major upgrade effectively uninstalls the existing installation of a product, and then installs the newer product version. The functionality that is required to install a minor upgrade is in the **Setup.exe** launcher. For more information, see Run-Time Behavior for Minor Upgrades. For more information, see Creating Minor Upgrades.
Automatic Upgrade Item	 ***Note*** • *The Automatic Upgrade Item option takes into account potential future changes in the product code and handles both major and minor upgrades. If you do not know the difference between upgrade types and/or do not care, consider choosing this upgrade option.* When you add an automatic upgrade item, the build engine determines which settings need to be populated into your installation to perform a successful upgrade of your previous installation. This option does not require any additional authoring of advanced settings on your part. For more information, see Configuring InstallShield to Automatically Determine the Upgrade Type.

Table 12-118 • Upgrades View Options (cont.)

Option	Description
Validate All Items	This option enables you to perform validation on the latest release or browse for a particular package to validate against a different release. For more information, see Validating Upgrades, Patches, and QuickPatch Packages. ***Note*** *• To validate a specific built upgrade item on demand, right-click the item and then click Validate Item.* ***Tip*** *• Whenever you make changes in the Upgrades view, remember to rebuild your package before performing validation.*

Common Tab

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

Project-specific differences are noted where appropriate.

The Common tab for the Upgrade Windows Installer Setup item in the Upgrades view contains global settings for both major and minor upgrades.

Small/Minor Upgrade Settings

Small updates and minor upgrades are functionally identical except the product version changes for a minor upgrade, but not a small update. A minor upgrade installs over an existing application while a major upgrade effectively uninstalls the existing installation of a product, and then installs the newer product version.

The functionality that is required to install a minor upgrade is in the **Setup.exe** launcher. For more information, see Run-Time Behavior for Minor Upgrades. To specify the behavior that should occur when **Setup.exe** detects that an earlier version of the product is present at run time on a target system, select the appropriate option:

Table 12-119 • Small and Minor Upgrade Options

Option	Description
Disable	When you choose this option, you are required to detect the upgrade scenario and ensure that the latest version of your setup is running in upgrade mode.
Don't prompt the user; just install the upgrade	This option automatically starts the setup in minor upgrade mode.

Table 12-119 • Small and Minor Upgrade Options (cont.)

Option	Description
Prompt	Choosing this option will prompt the "do you want to continue dialog." If you choose yes, the setup will begin in upgrade mode. If you choose no, you will cancel the setup in upgrade mode.

Major Upgrade Settings

When you are performing a major upgrade, you can select how you want the upgrade to proceed. You can choose from the following options:

Table 12-120 • Options for Performing a Major Upgrade

Option	Description
Completely uninstall old setup before installing new setup	This is the most reliable setting, however, it is the least efficient. It will first remove all the files, registry entries, shortcuts, and settings of the old setup. Then it will apply new data from the latest version of your setup
Install setup then remove unneeded files	This is the most efficient setting, but it requires that you follow certain authoring rules. This option will first install the setup, and then remove all unnecessary files, registry entries, shortcuts, and settings after installing the latest version of your setup. ***Caution*** • *The removal of unnecessary resources relies on component reference counts being accurate. Keep in mind that reference counts occur at the component level. Therefore, you should be careful when you delete a component and move the associated resource to a different component. You do not want to choose this option if you have moved existing resources to different components.*
Rollback all changes if removal of old files fails	In the event of upgrade failure, the machine will be returned to its previous good state. This option will undo the changes made by the uninstallation of a previous version and the installation of the latest version.

Note • *This setting controls the sequencing of the RemoveExistingProducts action in the Install Execute Sequence. For more details, see RemoveExistingProducts in the Windows Installer Help Library.*

Advanced Tab

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*

- *Transform*

Project-specific differences are noted where appropriate.

The Advanced tab presents specific settings for major and minor upgrades in addition to some shared settings.

Table 12-121 • Advanced Settings for the Upgrade Windows Installer Setup Item in the Upgrades View

Property	Description
Upgrade Code	This setting indicates the GUID that uniquely identifies the product family to which this product belongs. The upgrade code should be consistent across different versions and languages of a family of related products so that Windows Installer can use it to search for related versions of the product that are already installed. For more information on configuring this setting, see Setting the Upgrade Code. For more information on how the upgrade code is used in upgrades, see Major Upgrade vs. Minor Upgrade vs. Small Update.
On Upgrade	Specify the behavior that you want to occur if **Setup.exe** detects that an earlier version of your product is present on a target system. Available options are: • **Disable**—If an earlier version is present, **Setup.exe** does not display a message box to inform end users that an upgrade will occur. In addition, **Setup.exe** does not set the **REINSTALLMODE** and **REINSTALL** properties; therefore, these properties must be set through other methods. Otherwise, Windows Installer uses the cached copy of the .msi database and ignores the changes that are present in the newer version. • **Don't prompt the user; just install the upgrade**—If an earlier version is present, **Setup.exe** sets the **REINSTALLMODE** to **voums** and the **REINSTALL** property to **ALL**; as a result, the installation runs in upgrade mode. If an earlier version is not present, the installation behaves as a first-time installation. • **Prompt**—If an earlier version is present, **Setup.exe** displays a message box to inform end users that an upgrade will occur; the message box enables them to proceed with the upgrade or exit without upgrading. If an end user chooses to proceed, **Setup.exe** sets the **REINSTALLMODE** to **voums** and the **REINSTALL** property to **ALL**, and the installation runs in upgrade mode. This is the default value.

Table 12-121 • Advanced Settings for the Upgrade Windows Installer Setup Item in the Upgrades View (cont.)

Property	Description
Style	Specify how you want your major upgrade to proceed if the installation detects that an earlier version of your product is present on a target system. Available options are: • **Complete Uninstall Then Reinstall**—This is the most reliable setting; however, it is the least efficient. This option first removes all the files, registry entries, shortcuts, and settings of the old installation. Then it applies new data from the latest version of your installation. • **Install Then Remove Unused Files**—This is the most efficient setting, but it requires that you follow certain authoring rules. This option first installs the new version; then it removes all unnecessary files, registry entries, shortcuts, and settings that are not part of the new version. • **Install Then Remove Unused Files, with Rollback**—This is the most efficient setting, but it requires that you follow certain authoring rules. This option first installs the new version; then it removes all unnecessary files, registry entries, shortcuts, and settings that are not part of the new version. If the upgrade fails, the Rollback option returns the target system to its previous good state. This option undoes the changes that were made by the uninstallation of the earlier version and the installation of the latest version.  ***Caution*** • *Note for both of the options that first install and then removed unused resources: The removal of unnecessary resources relies on component reference counts being accurate. Reference counts occur at the component level. Therefore, you should be careful when you delete a component and move the associated resource to a different component. If you have moved existing resources to different components, do not choose either of these install-then-remove options.*

Major Upgrade Item: Common Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

A major upgrade is required when the product code for an installation differs from the product code of the installation that it needs to upgrade. For help in determining whether you need to change the product code, see Major Upgrade vs. Minor Upgrade vs. Small Update.

At run time, a major upgrade effectively uninstalls the previous version of your application before installing the newest version of your application.

The Common tab contains the most frequently used settings of a major upgrade item. For more detailed settings, use the Advanced tab.

Major Upgrade

Major upgrades use an upgrade code to detect previous versions of an application. The upgrade code for an installation groups that installation into a specific product family. If you want to target specific installations within that product family, you can configure the product version attribute of this upgrade item.

Table 12-122 • Options for Major Upgrade Items

Option	Description
Products Sharing My Upgrade Code	If you select this option for your major upgrade item, InstallShield sets the value of this setting on the Advanced tab to the placeholder value of **{00000000-0000-0000-0000-000000000000}**. InstallShield uses the appropriate upgrade code value instead of the placeholder value in the .msi package that it generates at build time.
Products Having Another Upgrade Code	If you select this option for your major upgrade item, you can either specify the upgrade code of the base package that this major upgrade item will target, or you can click the ellipsis button (...) to browse to the package. If you browse to the package, InstallShield identifies the upgrade code of the package that you select and uses it in this setting.

Product Version

Choose a particular product version to target or define a range. You can choose from the following selections:

Table 12-123 • Options Associated with Product Version

Option	Description
Any earlier version	To update the product on target systems if its version number is earlier than the product version of the currently open project, select this option. If you select this option for your major upgrade item, InstallShield sets the value of the Maximum Version setting on the Advanced tab to the placeholder value of *****ALL_VERSIONS*****. InstallShield uses the product version of the currently open project instead of the placeholder value in the .msi package that it generates at build time.

Table 12-123 • Options Associated with Product Version (cont.)

Option	Description
Within a specific range of versions (Minimum Version - Maximum Version)	To specify a range of product versions that you want the upgrade to target, select this option. Then, specify the minimum version in the left box, and the maximum version in the right box. The Version Range Inclusive check box lets you optionally include or exclude the specified version numbers in the range of target versions. For example, a minimum version of 1.00 and a maximum version of 4.00 results in an installation that upgrades any existing installation in the specified product family that has a version number between 1.00 and 4.00. If you select the Version Range Inclusive check box, the version search also checks for version 1.00 and version 4.00, and upgrades those versions if they are present.
Version Range Inclusive	To include both the minimum and maximum version numbers in the range of versions that the upgrade targets (instead of just the version numbers within the specified minimum-to-maximum range), select this check box. For example, a minimum version of 1.00 and a maximum version of 4.00 results in an installation that upgrades any existing installation in the specified product family that has a version number between 1.00 and 4.00. If you select the Version Range Inclusive check box, the version search also checks for version 1.00 and version 4.00, and upgrades those versions if they are present. To include only the minimum version number or only the maximum version number, but not both, use the Include Minimum Version setting or the Include Maximum Version setting on the Advanced tab. This check box is available if the **Within a specific range of versions** option is selected.
With a specific version	To target only a single previous version of the product, select this option, and then enter the appropriate version number.

Major Upgrade Item: Advanced Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *MSI Database*
- *Transform*

The Advanced tab exposes more advanced level configurations, in addition to the settings in the Common tab. Click on each setting in the properties grid to configure the following settings.

Table 12-124 • Advanced Properties for Major Upgrade Items

Property	Description
Upgrade Code	This setting indicates the GUID that uniquely identifies the product family to which this product belongs. The upgrade code should be consistent across different versions and languages of a family of related products so that Windows Installer can use it to search for related versions of the product that are already installed. If you select the **Products sharing my Upgrade Code** option on the Common tab for your major upgrade item, InstallShield sets the value of this setting on the Advanced tab to the placeholder value of **{00000000-0000-0000-0000-000000000000}**. InstallShield uses the appropriate upgrade code value instead of the placeholder value in the .msi package that it generates at build time. For more information on configuring this setting, see Setting the Upgrade Code. For more information on how the upgrade code is used in upgrades, see Major Upgrade vs. Minor Upgrade vs. Small Update.
Minimum Version	Enter the minimum product version that this upgrade should update. Version numbers should be in the format *aaa.bbb.ccccc*, where *aaa* represents the major version number, *bbb* represents the minor version number, and *ccccc* represents the build number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* is 65,535. ***Note*** • *Only the first three version fields are used when making product version comparisons. Windows Installer is unable to distinguish between product versions that differ only in their fourth field. To learn more, see Specifying the Product Version.* By default, each project includes an ISPreventDowngrade major upgrade item, which enables you to prevent end users from being able to install the current version of your product over a future major version of the same product. For this default major upgrade item, the default entry in the Minimum Version setting is the following placeholder value: *****ALL_VERSIONS***** InstallShield uses the product version of the currently open project instead of the aforementioned placeholder value in the .msi package that it generates at build time. To learn more about the ISPreventDowngrade support, see Preventing the Current Installation from Overwriting a Future Major Version of the Same Product.

Table 12-124 • Advanced Properties for Major Upgrade Items (cont.)

Property	Description
Include Minimum Version	This setting is useful when you are specifying upper and lower bounds for a range of product versions that your upgrade supports. To include the minimum version number in the range, select Yes.
Maximum Version	Enter the maximum product version that this upgrade should update. Version numbers should be in the format *aaa.bbb.ccccc*, where *aaa* represents the major version number, *bbb* represents the minor version number, and *ccccc* represents the build number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* is 65,535. ***Note*** • *Only the first three version fields are used when making product version comparisons. Windows Installer is unable to distinguish between product versions that differ only in their fourth field. To learn more, see Specifying the Product Version.* If you select the **Any earlier version** option on the Common tab for your major upgrade item, InstallShield sets the value of this setting on the Advanced tab to the following placeholder value: *****ALL_VERSIONS***** InstallShield uses the product version of the currently open project instead of the placeholder value in the .msi package that it generates at build time.
Include Maximum Version	This setting is useful when you are specifying upper and lower bounds for a range of product versions that your upgrade supports. To include the minimum version number in the range, select Yes.
Language	Specify a comma-separated list of languages that you want the upgrade to target or avoid targeting, and then use the Exclude Specified Languages setting to specify whether you want to exclude the languages that are configured in this setting. The language identifiers should be specified in decimal format. For example, specify **1033** for English (United States).
Exclude Specified Languages	Specify whether you want the installation to target or avoid targeting the languages that are identified in the Languages setting. Available options are: • **No**—If the language of the product on the target system matches one of the languages that are specified in the Language setting, and if the other requirements such as the version number are met, the installation performs an upgrade. • **Yes**—If the language of the product on the target system does not match any of the languages that are specified in the Language setting, and if the other requirements such as the version number are met, the installation performs an upgrade.

Table 12-124 • Advanced Properties for Major Upgrade Items (cont.)

Property	Description
Detect Only	Specify whether you want the installation to detect that an upgrade needs to occur without actually performing the upgrade. The default value is No. If Windows Installer determines that a related product is installed on the target system, it adds the product code of that related product to the property that is specified in the Detect Property setting.
Detect Property	If Windows Installer determines that a related product is installed on the target system, it adds the product code of that related product to the property that you specify in this setting. You can use the property that you specify in conditional statements to control the flow of the installation or even stop the installation altogether. Windows Installer sets this property when the FindRelatedProducts action runs in the installation sequences.
Only Remove Specified Features	If you want the installation to remove the entire earlier version of the product from target systems, leave this setting blank. If you want the installation to remove only some features of the earlier version of the product from target systems instead of removing the entire product, specify a comma-delimited list of the names of the features to be removed. Note that the installation does not remove any features if you specify formatted text that evaluates to an empty string.  ***Caution*** • *If you specify a value in this setting, the installation will not completely uninstall the earlier version of the product. Therefore, you will be left with two entries in Add or Remove Programs.*
Continue On Failure	Specify whether you want to proceed with the installation even if the uninstallation of the earlier version fails. Available options are: • **Yes**—The installation of the new version occurs, regardless of whether the uninstallation of the earlier version is successful. • **No**—The installation of the new version occurs only if the uninstallation of the earlier version is successful. This is the default value.
Migrate Feature States	Specify whether you want the installation to detect the feature states of the currently installed product and use these same states when applying the upgrade. This functionality may be useful if the feature tree in the new version of the product has not changed much from the original.

Minor Upgrade/Small Update Item Settings

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Minor and small upgrades are fundamentally the same except with minor upgrades, the product version changes.

You need to apply a small/minor upgrade any time you are upgrading a setup that has the same product code as the latest version of your setup. You are not required to change the product code of a setup unless you delete a component from a feature or you delete a feature from a setup.

Note • *If you move a component from one feature to another, you are essentially deleting that component from the first feature and therefore must change the product code.*

For upgrade scenarios in which the setup's product code has changed, you must apply a major upgrade.

This tab exposes the following settings:

Table 12-125 • Options for Minor and Small Upgrade Items

Option	Description
Setup to Upgrade	The functionality required to install a minor upgrade is in Setup.exe, which you must include in order for a minor upgrade to work properly. The setup specified at build time will also be used to perform upgrade validation. The build will also verify that the referenced setup can be updated using a minor upgrade. Note that the build will warn you if you do not include setup.exe when you build your release.
Release Flags	Specify release flags to include and exclude features depending on the type of release.

Automatic Upgrade Item Settings

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Note • *This is the preferred method for configuring upgrade settings. This option takes into account any potential future changes in the product code and handles both major and minor upgrades. If you do not know the difference between upgrade types and/or do not care, then choose this upgrade option.*

When you add an automatic upgrade item, the build engine determines which settings need to be populated into your setup to perform a successful upgrade of your previous setup. This option does not require any additional authoring of advanced settings on your part.

You can configure the following options for an automatic upgrade item in the Upgrades view:

Table 12-126 • Options for Automatic Upgrade Items

Option	Description
Setup to Upgrade	Enter the path to the setup project that needs to be upgraded or click the browse button to browse for it.
Release Flags	Associating a release flag with the automatic upgrade item will allow you to optionally exclude it from the build on a per release basis.

Releases View



Project • *The Releases view is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

After you have completely designed your project in InstallShield, you are ready to build a release for testing and, ultimately, distribution to end users. The Releases view contains settings that indicate how InstallShield should build your release.

When you build a release, InstallShield takes all of the information from your project and compiles it into a Windows Installer installation package (.msi file), merge module (.msm file), or an executable file and related files, depending on the project type, that are capable of installing your product onto any supported Windows platform.



Tip • *In some project types, you can also use the Release Wizard to configure the settings for your release.*

Product Configuration Settings



Project • *Product configurations are available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

For Basic MSI, InstallScript MSI, and Merge Module projects, every release that you build belongs to a product configuration. A product configuration provides a means for grouping together releases that share similar properties, such as the product name, product code, and package code.

Each product configuration has two tabs:

- General tab (available for Basic MSI, InstallScript MSI, and Merge Module projects)
- Multiple Instances tab (available for Basic MSI projects)

General Tab for a Product Configuration



Project • *The General tab for a product configuration is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

The General tab of each product configuration has the following settings, primarily for overriding other settings in the resulting releases:

Table 12-127 • Product Configuration Settings

Setting	Project Type	Description
Product Name	Basic MSI, InstallScript MSI, Merge Module	To override the product name in each release that you build under this product configuration, enter a new name. For information on how the product name is used, see Specifying a Product Name. Changing the product name in this setting does not affect any built releases. You must rebuild each release to see the changes reflected in the product name and in the project's folders and files.

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Product Version	Basic MSI, InstallScript MSI, Merge Module	To override the product version in each release that you build under this product configuration, enter the version number. The version must contain only numbers. It is typically in the format *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535. Note that although you can include the fourth field (*ddddd*), the installation does not use this part of the product version to distinguish between different product versions. To learn more, see Specifying the Product Version. If your release includes a **Setup.exe** file, the product version that you specify is displayed on the Properties dialog box for **Setup.exe**. For more information, see Customizing File Properties for the Setup Launcher.
Package Code	Basic MSI, InstallScript MSI, Merge Module	To override the package code that is entered in the General Information view, enter a new GUID. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. ***Note*** • *This package code is ignored if Yes is selected for the Generate Package Code setting.*
Generate Package Code	Basic MSI, InstallScript MSI, Merge Module	Specify whether you want InstallShield to generate a new package code every time that a release under this product configuration is built: • **Yes**—InstallShield generates a new package code at build time and includes it in the .msi package. The package code that is displayed in the Package Code setting in the General Information view does not change. • **No**—InstallShield does not generate a new package code at build time. If you enter a package code for the product configuration, that package code is used. If you do not specify a package code for the product configuration, InstallShield uses the package code that is set in the General Information view.

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Product Configuration Flags	Basic MSI, InstallScript MSI, Merge Module	Enter the release flags for the features, InstallShield prerequisites, and chained .msi packages that you would like to include in the releases that are associated with this product configuration. Separate multiple flags with a comma. Product configuration flags enable you to customize your installation by including or excluding certain features, InstallShield prerequisites, and chained .msi packages in each release that you build under this product configuration. Note that you can also specify release flags at the release level. For more information on filtering features, InstallShield prerequisites, and chained .msi packages, see Release Flags.
Subject	Basic MSI, InstallScript MSI, Merge Module	To override the value of the Subject setting in the General Information view for every release you build under this product configuration, enter the name of the product. The value that you enter is used on the Summary tab of the Properties dialog box that is displayed if you right-click the Windows Installer database and then click Properties. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Title	Basic MSI, InstallScript MSI, Merge Module	To override the value of the Title setting in the General Information view for every release you build under this product configuration, enter a new title. The value that you enter is used on the Summary tab of the Properties dialog box that is displayed if you right-click the Windows Installer database and then click Properties. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (**...**) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Template Summary	Basic MSI, InstallScript MSI, Merge Module	To override the Template Summary setting in the General Information view for a specific product configuration, specify the processor type and default language that your installation supports. List the processor type first, followed by your installation's default language, and separate them with a semicolon. If you have multiple entries in the language category, separate them with a comma. For example, if your installation runs only on Intel processors and English-based systems, enter **Intel;1033**. If your product runs on x64 processors and supports English and German, enter **x64;1033,1031**. For the language portion of this setting, use the number 0 if your installation is language neutral. Valid processor values include: • Intel • Intel64 • x64 Note that you can specify only one processor value. For more information, see Using the Template Summary Property. If the target machine does not meet the requirements that you specify for this setting, an error message is displayed and the installation exits.

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Architecture Validation	Basic MSI, Merge Module	Select the type of architecture validation that you want to use at build time. Available options are: • **Lenient**—This type of validation lets you build x86 and x64 .msi packages (as indicated by the Template Summary property), and mix x86 and x64 product files and custom action files in both of those types of packages. Lenient architecture validation does not trigger build errors or warnings if the architecture that the Template Summary property specifies does not match the architecture for one or more of the product files or the custom action files that are being included in the release. At build time, the ISRedistPlatformDependentFolder and ISRedistPlatformDependentExpressFolder path variables point to the x86 locations that contain x86 InstallShield custom action DLLs. InstallShield includes these x86 custom action DLLs in the build if your project includes support that requires these custom actions. Lenient is the default option.

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Architecture Validation (cont.)		• **Strict**—With this type of validation, InstallShield attempts to build a pure x86 or pure x64 .msi package, depending on whether the Template Summary property specifies Intel (for x86) or x64. Strict validation may trigger build errors if the architecture that the Template Summary property specifies does not match the architecture for one or more of the custom action files that are being included in the release. This type of validation may also trigger build warnings if the architecture that the Template Summary property specifies does not match the architecture for one or more of the product files that are being included in the release. If the Template Summary property specifies Intel, the ISRedistPlatformDependentFolder and ISRedistPlatformDependentExpressFolder path variables point to the x86 locations that contain x86 InstallShield custom action DLLs. However, if the Template Summary property specifies x64 and you are using strict validation, these path variables point to the x64 locations that contain x64 InstallShield custom action DLLs. If your project includes support that requires any of these custom actions, InstallShield adds the appropriate x86 or x64 versions of the DLLs. To learn more, see Selecting the Appropriate Type of Architecture Validation for Builds.

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Comments	Basic MSI, InstallScript MSI, Merge Module	To override the Summary Information Stream Comments setting in the General Information view for a specific product configuration, enter any comments about your product. A typical value for this setting is as follows: **This installer database contains the logic and data required to install MyProduct.** The value that you enter is used on the Summary tab of the Properties dialog box that is displayed if you right-click the Windows Installer database and then click Properties. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.
Schema	Basic MSI, InstallScript MSI, Merge Module	The schema version is an integer that identifies the minimum Windows Installer version that is required for the installation package. To override the Schema setting in the General Information view for a specific product configuration, enter the appropriate integer. For a minimum of Windows Installer 2.0, enter **200**. For a minimum of Windows Installer 3.0, enter **300**. For a minimum of Windows Installer 3.1, enter **301**. For a minimum of Windows Installer 4.5, enter **405**. If the end user's system has a Windows Installer version earlier than the minimum requirement that you specify for the Schema setting—for example, if you specify a schema value of 405 because your installation uses Windows Installer 4.5 features, but an end user has Windows Installer 3.1—the installation displays an error message and exits. The value that you enter for the Schema setting is used for the Page Count Summary property of your Windows Installer database.

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Product Code	Basic MSI, InstallScript MSI	To override the Product Code setting in the General Information view for a specific product configuration, enter a GUID that uniquely identifies this product. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. Since this code uniquely identifies your product, changing the product code after you have already distributed your release is not recommended. For more information, see Setting the Product Code in a Windows Installer–Based Project.
Upgrade Code	Basic MSI, InstallScript MSI	To override the Upgrade Code setting in the General Information view for a specific product configuration, enter a GUID that can be used for your product's upgrade code. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. The upgrade code is a GUID that identifies a related set of products. The Windows Installer uses a product's upgrade code when performing major upgrades of an installed product. The upgrade code, stored in the **UpgradeCode** property, should remain the same for all versions of a product. For more information, see Setting the Upgrade Code.

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Preprocessor Defines	Basic MSI, InstallScript MSI	To override the Preprocessor Defines setting on the Compile/Link tab of the Settings dialog box for a specific product configuration, specify any preprocessor definitions. Use the following format, with no spaces before or after equal signs or commas: **MYVARIABLE1=123,MYVARIABLE2=456** You can test those types of constants in your script by using #if and #ifdef statements that control the flow of the script. For example, type MYVARIABLE in this box, and the code in the following #ifdef loop will be executed: **#ifdef MYVARIABLE** **// Commands** **#endif** After you add or change a preprocessor definition in this box, you must compile your script for the addition or change to take effect.  ***Note*** • *Many Windows API functions are declared in the header file* **ISRTWindows.h**, *which is automatically included when you include* **Ifx.h** *in your script. You can prevent the automatic definition of Windows APIs by placing the preprocessor constant ISINCLUDE_NO_WINAPI_H in the Preprocessor Defines box on the Compile/Link tab of the Settings dialog box.*
MSI Package File Name	Basic MSI, InstallScript MSI	Specify the file name—without the period or the file extension—that InstallShield should use for the .msi file and—if applicable—the package definition file that it generates at build time. If this setting is blank, the product name is used.  ***Important*** • *If you want to be able to release minor upgrades or small updates to update your product, the previous and latest versions of your installation must have the same .msi package name. Attempting to perform a minor upgrade or a small update when the .msi file name has changed can lead to Windows Installer run-time error 1316.*

Table 12-127 • Product Configuration Settings (cont.)

Setting	Project Type	Description
Setup File Name	Basic MSI, InstallScript MSI	Specify the file name—without the .exe file extension—that InstallShield should use for the setup launcher file that it generates at build time. If this setting is blank, InstallShield uses the default value of *Setup*, and the setup launcher file is called **Setup.exe**.
Include Custom Action Help	Basic MSI, InstallScript MSI, Merge Module	Specify whether InstallShield should stream the contents of each of the custom action help files into the .msi file every time that a release in the selected product configuration is built. The path to a custom action's help file is indicated in the Help File Path setting in the Custom Actions and Sequences view (in Basic MSI and InstallScript MSI projects) or the Custom Actions view (in Merge Module projects). Selecting Yes is helpful if system administrators deploy your product to enterprise environments; they sometimes need to know what the custom actions do. If you select Yes, the following tasks occur at build time: • InstallShield adds the **ISCustomActionReference** table to the .msi file that is being built. The contents of all of custom actions' help files that are specified in the Custom Actions and Sequences view are streamed into the Description column of this table. • If your project includes any merge modules that contain custom actions, the contents of the help files that are specified in the Custom Actions view of the merge module projects are also included in the **ISCustomActionReference** table. For more information, see Documenting the Behavior of Custom Actions.

Multiple Instances Tab for a Product Configuration



Project • *The Multiple Instances tab for a product configuration is available in Basic MSI projects.*

For information on multiple-instance support for InstallScript projects, see Running an InstallScript Installation Multiple Times.

Windows Installer allows only one instance of a product code to be installed in the machine context and only one instance to be installed in each user context. Windows Installer 3.x and later includes support for a product code–changing transform. This type of transform—called an instance transform—enables the same .msi package to be used to install multiple instances of the same product in the same context because it changes the product code for each instance.

To create an installation that lets end users install multiple instances of your product, use the Multiple Instances tab. This tab is where you define different instances of your product and configure the properties that are associated with each instance. At build time, InstallShield creates an instance transform for each instance and streams the instance transforms into the .msi package. At run time, the installation typically displays an instance selection dialog that lets end users specify whether they want to install a new instance or maintain an existing one.



***Caution** • Creating an installation that lets end users install multiple instances of a product on the same machine and in the same context requires sophisticated authoring and serious commitment on the part of the installation developer. This functionality is recommended for only advanced installation developers.*

Release Settings

The settings for a release are organized by category on several different tabs in the Releases view:

- Build tab (available for Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, and Suite/Advanced UI projects)
- Setup.exe tab (available for Advanced UI, Basic MSI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects)
- Signing tab (available for Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, and Suite/Advanced UI projects)
- .NET/J# tab (available for Basic MSI and InstallScript MSI projects)
- Internet tab (available for Basic MSI, InstallScript, and InstallScript MSI projects)
- Events tab (available for Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, and Suite/Advanced UI projects)
- Windows App tab (available for Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, and Suite/Advanced UI projects)
-

Build Tab for a Release



***Project** • The Build tab is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The Build tab is where you configure how InstallShield should package your release.

Table 12-128 • Settings on the Build Tab

Setting	Project Type	Description
Release Location	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Enter the path to the top-level directory where your release will start to be built, or click the ellipsis button (...) to browse to the location.
Build Location	Advanced UI, Suite/Advanced UI	Enter the path to the top-level directory where your release will start to be built, or click the ellipsis button (...) to browse to the location.
Product Name	Advanced UI, Suite/Advanced UI	To override the product name that was specified in the General Information view with a new value for the selected release, enter the appropriate name. For information on how the product name is used, see Specifying a Product Name.
Product Version	Advanced UI, Suite/Advanced UI	To override the product version that was specified in the General Information view with a new value for the selected release, enter the appropriate version number. The version must contain only numbers. It is typically in the format *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535. Note that although you can include the fourth field (*ddddd*), the installation does not use this part of the product version to distinguish between different product versions. To learn more, see Specifying the Product Version. The product version that you specify is displayed on the Properties dialog box for **Setup.exe**. For more information, see Customizing File Properties for the Setup Launcher.
Suite GUID	Advanced UI, Suite/Advanced UI	To override the GUID that was specified in the General Information view with a new value for the selected release, enter the appropriate GUID. The Suite GUID uniquely identifies the Advanced UI or Suite/Advanced UI installation. To have InstallShield generate a different GUID for you, click the **Generate a new GUID** button ({...}) in this setting. Since this code uniquely identifies your Advanced UI or Suite/Advanced UI installation, changing the Suite GUID after you have already distributed your release is not recommended.

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
Media Format	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This read-only setting enables you to view the type of media selected for this release. You can set the media type only through the Release Wizard's Media Type panel; for InstallScript object projects, you cannot set the media type.
Compression	Basic MSI, InstallScript MSI, Merge Module	Specify whether the release should be compressed: • **Compressed**—InstallShield compresses all of your product's data files into .cab files. • **Uncompressed**—InstallShield does not compress your product's files into .cab files. As an alternative, you can use a custom compression method, which enables you to compress only the files that are associated with one or more features into .cab files. To set custom compression options, you must use the Release Configuration panel and the Custom Compression Settings panel in the Release Wizard. Note that when you use the Release Wizard to configure custom compression, InstallShield changes the Compression setting in the Releases view to a read-only value of **Custom (One Cab per Feature)** or **Custom (One Cab per Component)**. To change the compression from a custom method to the standard compressed method or the uncompressed method, you must use the Release Wizard again; you cannot do this from the Compression setting in the Releases view.  ***Note*** • *The output of the build process depends on the media type that you are building (such as CD-ROM, DVD-5, or Network Image), whether compression is used (as specified in the Compression setting), whether you are including a setup launcher (as specified in the Setup Launcher setting on the Setup.exe tab), and whether you are using disk spanning (as specified in the Disk Spanning setting, which is applicable to CD-ROM, DVD, and Custom media types).* *For example, if you select Compressed for the Compression setting, you select Yes for the Setup Launcher setting, and you are building a Network Image release, your data files are compressed into .cab files, and the .cab files are streamed into your* **Setup.exe** *file. If you select Uncompressed for the Compression setting, you select No for the Setup Launcher setting, and you are building a CD-ROM release that fits on one CD-ROM and uses automatic disk spanning, your data files are left uncompressed in a subfolder of the folder that contains the .msi file.*

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
Cab Optimization Type	Basic MSI, InstallScript MSI, Merge Module	If Compressed or one of the custom options is selected for the Compression setting, specify the type of compression that InstallShield should use when building this release's .cab files. Available options are: • **LZX**—InstallShield uses LZX compression to compress your product's data files into .cab files. This option results in the smallest .cab files; however, it also takes the most time to extract the data from these .cab files at run time. • **MSZIP**—InstallShield uses MSZIP compression to compress your product's data files into .cab files. This is the default option. • **None**—InstallShield does not use any compression when creating the .cab files.  ***Important*** • *Using compression generally decreases the size of your compressed files, but the build process may take more time to complete. Depending on the number and size of the files being compressed, the LZX compression and the build may take hours to complete. Therefore, if you select the LZX option, it is recommended that your build machine have the latest hardware to minimize the time that it takes for the build to complete.*
Compress Script	InstallScript, InstallScript Object	Specify whether the compiled script file (.inx file) is placed in a cabinet file (Yes), or it is placed uncompressed in the Disk1 disk image folder (No).

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description

Compiler Preprocessor Defines	InstallScript, InstallScript Object, Suite/ Advanced UI	Optionally specify any preprocessor variable definitions. Preprocessor variable definitions that are specified here apply only to the current release; they are not used when compiling the script for other releases. Use the following format, with no spaces before or after equals signs or commas:
		
		MYVARIABLE1=123,MYVARIABLE2
		
		Such variables can be tested in the script by #if and #ifdef statements that control the flow of the script. If you are creating an InstallScript object, enter **IFX_OBJECTS**.
		
		Entering the name of a preprocessor variable for this setting defines the variable. For example if you enter **MYVARIABLE** for this setting, the script commands in the following #ifdef loop are executed:
		
		`#ifdef MYVARIABLE`
		`  // Commands`
		`#endif`
		
		After you add or change a preprocessor variable definition for this setting, you must compile your installation for the addition or change to take effect.

Disk Spanning	Basic MSI, InstallScript MSI	This read-only setting indicates how your installation is spread across different disks. You can select a different option through the Disk Spanning Options panel in the Release Wizard. Available options are: • **Automatic**—InstallShield automatically splits your release files over as many disks as necessary. • **Custom - Enforce disk size**—InstallShield displays a warning and aborts the build if any feature's files are larger than the disk size. • **Custom - Do not enforce disk size**—InstallShield builds the release to span as many disks as necessary. This setting applies to the CD-ROM, DVD, and Custom media types. ***Important*** • *Multi-disk installations cannot be run from a non-removable media (for example, from a hard drive). If you want to test an installation that spans disks, you need to put the installation on your target media. If you do not, the installation will fail because of a limitation of the Windows Installer.*

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
File Name Format	Basic MSI, InstallScript MSI, Merge Module	Select the file name format that should be used to determine how the paths of your files are stored in the .msi package. InstallShield stores the option that you specify in your .msi package's Summary Information Stream. If your installation will be distributed on media that does not support long file names, such as a UNIX server, select the Short File Names option.
Platform(s)	InstallScript, InstallScript Object	If the release is specific to one or more platforms, use this setting to indicate the platforms: click the value of this setting, and then click the ellipsis (...) button to specify the appropriate platforms. If the platform specified for a component does not match one of the platforms that is selected for this setting, the component is not included in the release. The default value for this setting is Use Project Settings. This value indicates that the release supports the platforms that are specified at the project level.
Layout	InstallScript	This read-only setting indicates whether the project's files are stored in cabinet files or placed uncompressed in the disk image. You can modify this setting through the Media Layout panel in the Release Wizard. Available options are: • **Cabinet File(s)**—InstallShield places all data files in one or more data cabinet files at build time. • **CD-ROM Folder(s)**—InstallShield places all data files in one or more CDROM folders at build time. • **Custom**—InstallShield places some data files in one or more CD-ROM folders and some in one or more data cabinet files at build time.
Data Languages	Basic MSI, InstallScript MSI, Merge Module	If you want to include certain components and exclude others based on the language that is selected for each component, click the ellipsis button (...) in this setting and specify the appropriate languages. If the language specified for a component does not match one of the languages that is selected for this setting, InstallShield does not include the component in the release. By default, releases are language independent; that is, none of the project's components are excluded from the release.

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
UI Languages	Advanced UI, Basic MSI, InstallScript MSI, Merge Module, Suite/Advanced UI	This setting lets you specify which user interface languages you want to include in a release. To select the appropriate languages, click the ellipsis button (...) in this setting. Note that if a language is not selected in the Setup Languages setting in the General Information view of the project, it is not listed as one of the available languages for the UI Languages setting.
Language(s)	InstallScript, InstallScript Object	Use this setting if you want to include certain components and exclude others based on the language that is selected for each component. This setting also lets you specify which user interface languages you want to include in a release. If the language specified for a component does not match one of the languages that is selected for this setting, InstallShield does not include the component in the release. In addition, if a UI language that is included in the project does not match one of the languages that is selected for this setting, InstallShield does not include the UI strings in the release. By default, releases are language independent; that is, none of the project's components or UI strings are excluded from the release. Note that if a language is not selected in the General Information view of a project, it is not listed as one of the available languages for the Language(s) setting.
Default Language	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Suite/ Advanced UI	To override the release's default project language that is configured in the General Information view or the String Editor view, select the appropriate default user interface language for your installation. For more information, see Setting the Default Project Language.
Languages Dialog	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Suite/ Advanced UI	Specify whether you want your installation to display the language selection dialog when the installation is run with a full user interface. ***Tip*** • *For Basic MSI projects—If you want the language selection dialog to be displayed, a* **Setup.exe** *setup launcher is required. To learn more, see Creating a Setup Launcher.*

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
Build UTF-8 Database	Basic MSI, InstallScript MSI, Merge Module	Specify whether you want the Windows Installer database, along with any instance or language transforms, to be built using the UTF-8 encoding. The UTF-8 encoding supports characters from all languages simultaneously, enabling you to mix and match, for example, Japanese and German, or Russian and Polish, both in text shown to end users and in file names and registry keys. These mixed languages work correctly regardless of the current language of the target system.  ***Important*** • *As documented in Microsoft KB 979849 the Windows Installer does not fully support UTF-8 databases; therefore, some scenarios result in user interface issues. For example, if an end user runs the installation with a basic user interface (by using the* `/qb` *command-line option) or uninstalls the product from Add or Remove Programs, the Windows Installer uses very small fonts to display the user interface text in a UTF-8 database.* If you specify No, InstallShield creates an ANSI database when you build your release. This option does not let you mix characters from languages in different code pages. The default value for this setting is No.
Previous Package	Basic MSI, InstallScript MSI, Merge Module	Enter the fully qualified path to a previous release (.msi file for an installation project, or .msm file for a merge module project) to minimize the size of future patch packages. This setting corresponds to the Patch Optimization setting in the Release Wizard.  ***Note*** • *If your package uses dynamic file linking, it is recommended that you specify a previous package in this setting so that the file keys are consistent across releases.* For more information, see Upgrade Considerations.

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
Generate File Hash Values	Basic MSI, InstallScript MSI, Merge Module	Indicate whether you want InstallShield to populate the **MsiFileHash** table for every unversioned file in your build. Available options are: • **Yes**—InstallShield populates the **MsiFileHash** table for every unversioned file in your build. • **No**—InstallShield does not populate the **MsiFileHash** table. If you select No, InstallShield builds any entries that are found in the project's **MsiFileHash** table (populated using the Direct Editor). If you have already populated the **MsiFileHash** table for a particular file, the build uses that information instead of generating the information at build time. For more information, see MsiFileHash Table in the Windows Installer Help Library.
Shallow Folder Structure	Basic MSI, InstallScript MSI	If you want InstallShield to create the .msi file and related files directly in the location that is specified in the Release Location setting, without any of the subfolders, select Yes. To build the release if the release location is in <ISProjectDataFolder> or <ISProjectFolder>, click Build Tables & Refresh Files on the Build menu.
Generate Autorun.inf	Basic MSI, InstallScript MSI	Select Yes if you are distributing your installation on a CD-ROM or DVD-ROM and you want to support the AutoPlay feature. InstallShield creates a text file called **Autorun.inf**, which contains the instructions to autoplay your installation, in the root of your disk images folder. You can edit this file to add additional AutoPlay options or to pass command-line parameters to **MsiExec.exe**, **Setup.exe**, or **Update.exe**.

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
Release Flags	Advanced UI Basic MSI, InstallScript MSI, Suite/Advanced UI	Release flags enable you to customize your installation by including or excluding certain items in each release. Enter the flags that you would like to include in this release. Separate multiple flags with a comma.  ***Project*** • *For Basic MSI and InstallScript MSI projects—Once you have assigned release flags to your features, InstallShield prerequisites, and chained .msi packages, you can create a release that includes features, InstallShield prerequisites, and chained .msi packages based on those flags. By default, all features, InstallShield prerequisites, and chained .msi packages are included in a release. Once you specify a flag in either the Releases view or the Release Wizard, only unflagged items and items that contain the specified release flag are included in your installation.* *For Advanced UI and Suite/Advanced UI projects—Once you have assigned release flags to features and packages in the Advanced UI or Suite/Advanced UI project, you can create a release that includes features and packages based on those flags. By default, all features and packages are included in a release. Once you specify a flag in the Releases view, only unflagged items and items that contain the specified release flag are included in your installation.*  ***Note*** • *If a release does not have release flags, it will include all applicable items that have release flags. To include only unflagged items, specify a flag that does not exist. For example, you might use* ***NoFlags****. This way, only unflagged items are built into a release.* *For Basic MSI and InstallScript MSI projects—Note that you can also specify release flags at the product configuration level. For more information, see Product Configuration Flags vs. Release Flags.*
Use Path Variable Test Values	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you used test values for any of your path variables, select Yes to set those variables to their actual values at this time.

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
Path Variable Overrides	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	To override a path variable in your project at build time for the selected release, click the ellipsis button (...) in this setting, which launches the Path Variable Overrides dialog box. This dialog box lets you select one or more path variables that you want to override. InstallShield adds a new path variable setting for each path variable that you select. Configure each path variable setting as necessary. Note that you can override user-defined path variables, environment variables, and registry variables that are configured in the Path Variables view; however, you cannot override predefined path variables such as *<WindowsFolder>*. To learn more, see: • Overriding the Value of a Custom Path Variable for a Release • Using Path Variables
Path Variable Overrides Item	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	Specify the new value of the path variable that you want InstallShield to use at build time for the selected release. The default value is the setting that is configured in the Path Variables view. InstallShield uses the new value that you specify in this setting to override the value that is set in the Path Variables view. To delete the path variable override, click the Delete button in this setting.  ***Tip*** • *You can refer to other path variables in the defined value by enclosing the referenced path variable name in angled brackets. For example, if you have a path variable called* `MyRoot` *with a value of* **C:**, *you can refer to it in a path variable definition for another variable, such as* `Games`. *The actual path for the* `Games` *variable might be* **C:\Programs\GameFiles,** *but you can define* `Games` *as* **<MyRoot>\Programs\GameFiles**. *However, if you attempt to self-reference a path variable, the literal string is used instead. For example, defining* `Games` *as* **<MyRoot>\Programs\<Games>** *actually results in* `Games` *defined as* **C:\Programs\<Games>**. To learn more, see: • Overriding the Value of a Custom Path Variable for a Release • Using Path Variables

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
Features	InstallScript, InstallScript Object	This read-only setting indicates which features are included in the built release. You can modify the value of this setting from the Features panel in the Release Wizard. • **Use the "Include in Build" feature property**—If this option is selected, each feature whose Include In Build setting is set to Yes is included in the built release, and each feature whose Include In Build setting is set to No is not included. • **Specify the features to be included directly**—Specify for each feature whether the feature is included in the built release.
Specify Skin	InstallScript	Select the dialog skin that is applied to this release. The default value, <Use Project Setting>, uses the skin that is selected in the Skins folder of the Dialogs view; selecting any other value for this setting overrides the Skins folder selection.
Differential Media	InstallScript	Specify whether the current release (the release that is selected in the Releases view) is a differential release—that is, a release that contains only those files that were absent from one or more of a specified set of existing releases—or a full release, which contains all your product's files, so that your product can be installed on a system on which no version of your product is currently installed. For more information, see Differential vs. Full Releases.
Support Version(s)	InstallScript	This setting is applicable only if No is selected for the Differential Media setting. Optionally enter a semicolon-delimited list of version numbers (for example, 1.2.3;1.2.4) of the earlier versions of your product to which this release can be applied as an update. If you leave this setting blank, the installation can be run on a system on which any earlier version, or no version, of your product is currently installed.
Object Difference	InstallScript	This setting is applicable only if Yes is selected for the Differential Media setting. Select the conditions for including InstallShield objects in your differential release. • **Include All**—The differential release includes all objects that would be included in the equivalent full update release. • **Exclude All**—The differential release does not include any objects. • **Include If Changed**—The differential release includes those objects that would be included in the equivalent full update release and that are not found in, or are different from, the corresponding object in at least one of the comparison releases.

Table 12-128 • Settings on the Build Tab (cont.)

Setting	Project Type	Description
Keep Unused Directories	Basic MSI, InstallScript MSI, Merge Module	Specify whether you want InstallShield to remove unused directories from the **Directory** table of the .msi file when you build this release. Available options are: • **No**—If a directory that is listed in the Directory column of the **Directory** table is not referenced in any known location in the .msi file, InstallShield removes it from the **Directory** table of the .msi file that it creates at build time. For Basic MSI and InstallScript MSI projects, this occurs after any merge modules are merged, but only directories that are present in the .msi file are removed; therefore, if a merge module contains new unused directories in its **Directory** table, the new unused directories are added to the installation. • **Yes**—InstallShield does not remove any directories from the **Directory** table of the .msi file that it creates at build time. The default value is No.  ***Note*** • *Under some conditions, predefined directories cannot be resolved, causing an installation to fail. Removing unused directories from the **Directory** table enables you to avoid unnecessary failures. Therefore, it is recommended that you select No for this setting.*
Hide Add/ Remove Panel Entry	InstallScript MSI	Specify whether you want to hide your product's entry in Add or Remove Programs in the Control Panel: • **No**—Your product's entry is displayed on the target machine in Add or Remove Programs. • **Yes**—Your product's entry is not displayed on the target machine in Add or Remove Programs. The end user cannot use Add or Remove Programs to remove, modify, or view support information for your product. To learn about how to indicate which information should be displayed for your product—including how to hide buttons in the Add or Remove Programs—see General Information View Settings.

Setup.exe Tab for a Release



Project • *The Setup.exe tab is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

- *Suite/Advanced UI*

For more information on including Windows Installer redistributables in installations, see Adding Windows Installer Redistributables to Projects.

The Setup.exe tab is where you configure settings about your **Setup.exe** file. It is also where you specify whether you want to include redistributables for Windows Installer 3.1 or earlier. For more information on Windows Installer redistributables, see Adding Windows Installer Redistributables to Projects.

Table 12-129 • Settings on the Setup.exe Tab

Setting	Project Type	Description
Setup Launcher	Basic MSI, InstallScript MSI	Specify whether you want to create a **Setup.exe** setup launcher. To learn about scenarios that require a setup launcher, see Creating a Setup Launcher.
Executable File Name	Advanced UI, Suite/Advanced UI	Specify the file name—without the .exe file extension—that InstallShield should use for the setup launcher file that it generates at build time. If this setting is blank, InstallShield uses the default value of *Setup*, and the setup launcher file is called **Setup.exe**.
Single .exe File Name	InstallScript	If you want InstallShield to build a self-extracting executable file that runs your installation, specify its file name (including the file extension). If you specify a file name, InstallShield creates a self-extracting executable file and adds it to the Package subfolder of the folder that is specified in the Release Location setting on the Build tab for this release. If you leave this setting empty, InstallShield does not create a self-extracting executable file at build time.
Setup.exe Icon File	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	To use your own icon for the **Setup.exe** file, specify the fully qualified name of the file that contains the icon. To specify a file, type an absolute path or a path that is relative to a path variable, or click the ellipsis button (...) to browse to the file from within the Change Icon dialog box. By default, the icon with index 0 is used; to specify a different icon, either select an icon in the Change Icon dialog box or append the icon's index or resource ID (preceded by a minus sign) to the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100. If you leave this setting blank, InstallShield uses a default icon for your **Setup.exe** file.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description

MSI Command-Line Arguments	Basic MSI, InstallScript MSI	Indicate any command-line arguments that you want to pass from the setup launcher to Windows Installer. InstallShield writes the value that you enter in the **Setup.ini** file.
		`[Startup]`
		`CmdLine=Your Value`
		
		Tip • *To set a property, enter* **MYPROPERTY="My Value"** *for this setting.*

Setup Command Line	InstallScript	Enter any command-line parameters that you want to pass to **Setup.exe** when end users launch the installation.
Generate Package Definition File	Basic MSI, InstallScript MSI	Specify whether you want InstallShield to create a package definition file (.pdf) that enables end users to run your installation as an SMS job. If you select Yes, InstallShield creates a version 2.0 .pdf.
Required Execution Level	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	Use the Required Execution Level setting to specify the minimum level that is required by your project's **Setup.exe** file for running on Windows Vista and later platforms. The available options are: • **Administrator**—**Setup.exe** requires administrative privileges to run. Administrators must authorize it; non-administrators must authenticate as an administrator. • **Highest available**—**Setup.exe** prefers administrative privileges. Administrators must authorize it; non-administrators run it without administrative privileges. This is the default option for InstallScript and InstallScript MSI projects. • **Invoker**—**Setup.exe** does not require administrative privileges, and all users can run it without administrative privileges. **Setup.exe** does not display any UAC messages prompting for credentials or for consent. This is the default option for Advanced UI, Basic MSI, and Suite/Advanced UI projects. For Advanced UI, InstallScript, InstallScript MSI, and Suite/Advanced UI projects, and for Basic MSI projects if Yes is selected for the Setup Launcher setting, InstallShield embeds an application manifest in the **Setup.exe** launcher. This manifest specifies the selected execution level. Operating systems earlier than Windows Vista ignore the required execution level. If No is selected for the Setup Launcher setting for a Basic MSI project, InstallShield does not embed the Windows application manifest in a **Setup.exe** launcher.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
Required Execution Level (cont.)		The benefit of elevating the required execution level in Basic MSI projects is that privileges can be elevated only once if necessary to run **Setup.exe**, and that these privileges can be carried over to all of the installation's InstallShield prerequisites and the Execute sequence of the .msi package without requiring multiple prompts for approval. Thus, if two of your InstallShield prerequisites require administrative privileges, for example, you can change this setting to Administrator, and then end users are prompted only once during the installation, before Windows Installer runs the **Setup.exe** file. A similar benefit exists for Advanced UI and Suite/Advanced UI projects, where privileges can be elevated only once if necessary to run **Setup.exe**, and that these privileges can be carried over to all of the installation's Advanced UI or Suite/Advanced UI packages without requiring multiple prompts for approval. Note, however, that if you elevate the privileges and also launch the application at the end of the installation, the elevated privileges are carried over to the application. In most cases, running an application with elevated privileges on Windows Vista and later platforms is discouraged. For more information, see Minimizing the Number of User Account Control Prompts During Installation.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
Advertise If Prerequisites Are Elevated	Basic MSI, InstallScript MSI	In the following common scenario, you may want to advertise your .msi file to help end users avoid a second User Account Control (UAC) prompt during your product's installation on Windows Vista and later systems: • Your installation includes one or more InstallShield prerequisites that require elevation. • In addition, your .msi file requires elevation. (That is, the Require Administrative Privileges setting in the General Information view is set to the default value of Yes.) If this scenario does not apply to your installation, it is recommended that you leave the default value of No for the Advertise If Prerequisites Are Elevated setting because setting it to Yes would not avoid a second UAC prompt. The Advertise If Prerequisites Are Elevated setting applies to installations that are run on Windows Vista and later systems. Installations that are run on earlier versions of Windows ignore this setting. Valid options are: • **Advertise: Silent**—Indicates that if InstallShield prerequisites in the installation are successfully installed with elevated privileges, the .msi file should be advertised and run silently (without a user interface). Then the main part of the installation does not require an additional UAC prompt for consent or credentials. • **Advertise: Full UI**—Indicates that if InstallShield prerequisites in the installation are successfully installed with elevated privileges, the .msi file should be advertised and run with a full user interface. Then the main part of the installation does not require an additional UAC prompt.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
(cont.)		• **No**—Indicates that the .msi file should not be advertised. When end users run the installation, one or more UAC prompts may be displayed to install the InstallShield prerequisites. If the .msi file also requires elevation, an additional UAC prompt may be displayed before the main part of the installation occurs. ***Note*** • *The package must support advertisement in order for either of the advertise options to work. Advertisement is not instantaneous, and it adds extra delays to the installation. In addition, unexpected behavior may occur if the end user clicks Cancel after advertisement but before the main part of the installation has finished. For example, advertised shortcuts for your product may appear on the desktop before the main installation begins, and a confused user canceling the main installation may leave your package advertised but not fully installed. Therefore, in some cases, it may be better to leave this setting as No to allow the second UAC prompt and avoid product advertisement.* In some cases, the .msi file is not advertised, and as a result, the second UAC prompt is displayed. For more information, see Specifying Whether a Product Should Be Advertised If Its InstallShield Prerequisites Are Run with Elevated Privileges.
Include MSI Engine	Basic MSI, InstallScript MSI	Specify whether you want to include redistributables for Windows Installer 3.1earlier. ***Note*** • *This setting applies to redistributables for Windows Installer 3.1. For information on different methods of adding various versions of Windows Installer redistributables to a project, see Adding Windows Installer Redistributables to Projects.*

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
MSI Engine Location	Basic MSI, InstallScript MSI	 ***Note*** • *This setting applies to redistributables for Windows Installer 3.1. For information on different methods of adding various versions of Windows Installer redistributables to a project, see Adding Windows Installer Redistributables to Projects.* Specify where the Windows Installer engine installers should be located. Available options are: • **Copy from Source Media**—Leave the selected Windows Installer engine installers on the root of the source media. This option is not available for Web media types or Network Image media types with all of your files compressed into **Setup.exe**. • **Extract Engine from Setup.exe**—Compress the selected Windows Installer engine installers into **Setup.exe**, to be extracted at run time. • **Download Engine from the Web**—Download the Windows Installer engine installer (if necessary) from a URL that you specify. If you select this setting, be sure to set the engine URL settings to the appropriate Web locations, or leave the default value.
MSI 3.1 Engine URL	Basic MSI, InstallScript MSI	This setting specifies the location from which the setup launcher downloads the Windows Installer 3.1 engine, if needed. This setting is ignored unless the MSI Engine Location setting for the current release is set to Download Engine from the Web. Do not specify the file name in the URL.  ***Note*** • *The default URL (http://www.installengine.com/Msiengine30) is a site maintained by Flexera Software for your convenience. The 3.1 engine is available for download from this location.*

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
Delay MSI Engine Reboot	Basic MSI, InstallScript MSI	 ***Note*** *• This setting applies to redistributables for Windows Installer 3.1 and earlier. For information on different methods of adding various versions of Windows Installer redistributables to a project, see Adding Windows Installer Redistributables to Projects.* Specify whether you want to postpone any reboot associated with installing or updating the Windows Installer engine on the target system until after your installation has completed. Select Yes to postpone the reboot, if one is necessary. Select No to allow the system to reboot, if necessary, immediately after the Windows Installer engine has been installed or updated and before performing your installation.
Cache MSI Locally	Basic MSI, InstallScript MSI	Specify whether the .msi file and other installation files for the current build should be cached on the target system. • **Yes**—Cache the .msi file and other installation files on the target system for use with application maintenance and repair. The Cache Path setting for the current release specifies where files should be cached. • **No**—Do not cache the .msi file and other installation files on the target system.  ***Note*** *• This setting is enabled for releases that do not have the .msi file available in the same folder as the* **Setup.exe** *file on the target system.*
Cache Path	Basic MSI, InstallScript MSI	Specify where the cached .msi file and other cached installation files should be stored on the end users's system. You can enter a hard-coded value such as **C:\CachedFiles**, but it is recommended that you cache files in a path using a destination variable selected from the list. The default value is **[LocalAppDataFolder]Downloaded Installations**.  ***Note*** *• This setting is used only if the Cache MSI Locally setting for the current release is set to Yes.*
Media Password	InstallScript	Enter the password that end users must enter in order to run the installation. Leave this setting empty if you do not want to password-protect your installation.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
Show Password Dialog	InstallScript	This setting is applicable only if you specify a non-null password in the Media Password setting. Select Yes to set the *SHOW_PASSWORD_DIALOG* system variable to a non-zero value, which executes the password-checking code in the OnCheckMediaPassword event handler function's default code. If you select No, you must enter your own code in the script to request the password from the end user and check it by calling **FeatureValidate**.
Small Initialization Dialog	Basic MSI, InstallScript, InstallScript MSI	Specify whether you want to display a small initialization dialog when end users run this release. When the installation initializes, by default, an initialization dialog is displayed. The standard initialization dialog is the same size as the Welcome dialog and contains initialization text, a progress bar, and a Cancel button. The small initialization dialog contains initialization text, progress and a Cancel button; however, it is smaller, it does not contain a bitmap image, and its background is the window color of the end user's system.
Init Dialog Product Name	InstallScript	Optionally enter the product name to display in the setup initialization dialog. If you leave this setting empty, InstallShield uses the product name that you specify in the General Information view.
Minimum Initialization Time	Basic MSI, InstallScript, InstallScript MSI	Specify the minimum number of seconds that the installation should display the initialization dialog when end users run this release. InstallShield uses the value that you specify for this setting as the value of the SplashTime keyname in the **Setup.ini** file. When the installation initializes, by default, an initialization dialog is displayed. The splash screen, if provided, is also shown at this time. If you specify a minimum initialization time in this setting, the initialization dialog and splash screen are shown for at least the specified number of seconds. If initialization takes longer than the time specified, the dialog and splash screen are still displayed until the installation completes initialization. If initialization takes less time than the time specified, the installation continues to display the dialog and splash screen until the specified time has elapsed, and then the installation continues.
Password Protect Launcher	Advanced UI, Basic MSI, InstallScript MSI, Suite/Advanced UI	To password-protect your installation, select Yes, and then type a password for the Launcher Password setting.  ***Project*** • *In Basic MSI and InstallScript MSI projects, this setting is available only if you select Yes for the Setup Launcher setting.*

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
Launcher Password	Advanced UI, Basic MSI, InstallScript MSI, Suite/Advanced UI	Type a password to protect your installation. Passwords are case-sensitive. This setting is used only if you select Yes for the Password Protect Launcher setting.  ***Project*** • *In Basic MSI and InstallScript MSI projects, this setting is available only if you select Yes for the Setup Launcher setting.*
Use Custom Version Properties	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	Specify whether you want to override the default copyright notice and file description for **Setup.exe** with your own copyright notice and file description. If you select Yes, enter your own information in the Launcher Copyright setting and the File Description setting. The copyright and description are displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. To learn more, see Customizing File Properties for the Setup Launcher.
Launcher Copyright	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	If you want to override the default copyright notice for **Setup.exe** with your product's copyright notice, enter your product's copyright notice. Note that you must also select Yes in the Use Custom Version Properties setting. The copyright is displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. To learn more, see Customizing File Properties for the Setup Launcher.
File Description	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	If you want to override the default file description for **Setup.exe** with your own description, enter the appropriate description. Note that you must also select Yes in the Use Custom Version Properties setting. The description is displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. To learn more, see Customizing File Properties for the Setup Launcher.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
Version	Advanced UI, Suite/Advanced UI	If you want to override the default product version and file version for **Setup.exe** with your own version number, enter the appropriate version number. Note that you must also select Yes in the Use Custom Version Properties setting. The product version and file version are displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. Note that the file version always contains four fields. If you specify fewer than four fields for your product version, the remaining fields are filled with zeros. For example, if you specify a product version of **1.1**, the file version that is used in the version resources of **Setup.exe** is **1.1.0.0**. To learn more, see Customizing File Properties for the Setup Launcher.
Company	Advanced UI, Suite/Advanced UI	If you want to override the default company name for **Setup.exe** with your own company name, enter the appropriate name. Note that you must also select Yes in the Use Custom Version Properties setting. The company name is displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. To learn more, see Customizing File Properties for the Setup Launcher.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
InstallShield Prerequisites Location	Basic MSI, InstallScript, InstallScript MSI	Specify where the InstallShield prerequisites that are selected in the Redistributables view (in Basic MSI and InstallScript MSI projects) or in the Prerequisites view (in InstallScript projects) should be located. ***Project*** • *The available options differ, depending on which project type you are using.* • **Follow Individual Selections**—Use the locations that are specified for each individual InstallShield prerequisite's properties in the Redistributables or Prerequisites view. This option is available for Basic MSI, InstallScript, and InstallScript MSI projects. • **Download From The Web**—Download all of the InstallShield prerequisite files included in your project (if necessary) from the URL specified in the InstallShield prerequisite (.prq) file for each prerequisite. This option overrides the locations that are specified in the Redistributables or Prerequisites view for each InstallShield prerequisite's properties. This option is recommended if your installation will be downloaded from the Internet and you want to minimize the package size and download time. An InstallShield prerequisite will not be downloaded if the correct version is already present on the target machine. This option is available for Basic MSI, InstallScript, and InstallScript MSI projects. • **Extract From Setup.exe**—Compress the InstallShield prerequisite files into **Setup.exe**, to be extracted at run time, if necessary. This option overrides the locations that are specified in the Redistributables view for each InstallShield prerequisite's properties. Select this option if the entire installation must be self-contained in **Setup.exe**. Note that the Download From The Web option results in smaller installations and shorter download time; however, the Extract From Setup.exe option provides for a completely self-contained installation. This option is available for Basic MSI and InstallScript MSI projects.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
InstallShield Prerequisites Location (cont.)		• **Copy From Source Media**—Store the InstallShield prerequisite files on the source media. This option overrides the locations that are specified in the Redistributables view for each InstallShield prerequisite's properties. Use this option if the installation will be run uncompressed from fixed media—CD, DVD, or local network. This option is available for Basic MSI and InstallScript MSI projects. • **Include with Media**—Store the InstallShield prerequisite files on the source media or in **Setup.exe**, depending on how you configure the settings for the release. This option overrides the locations that are specified in the Prerequisites view for each InstallShield prerequisite's properties. This option is available for InstallScript projects. Note that if an InstallShield prerequisite is added to a project as a dependency of another prerequisite, the location for the prerequisite dependency follows the location setting of the prerequisite that requires it.  ***Tip*** • *If you select the Extract From Setup.exe option, the Copy From Source Media option, or the Include with Media option and then build a release that includes an InstallShield prerequisite that is not available on your computer, one or more build errors are generated for every file that the prerequisite requires. To avoid these build errors, use the Redistributables view or the Prerequisites view to either download the InstallShield prerequisite from the Internet to your computer or remove it from your project before building the release.* To learn more, see Specifying the Run-Time Location for InstallShield Prerequisites at the Release Level.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
Package Location	Advanced UI, Suite/Advanced UI	Specify the run-time location for the packages that are configured in the Packages view. Available options are: • **Copy From Source Media**—Store the packages on the source media. This option overrides the run-time locations that are specified for each individual package's Location setting in the Packages view. If the Advanced UI or Suite/Advanced UI installation will be run uncompressed from fixed media—CD, DVD, or local network—select this option. • **Extract From Setup.exe**—Compress the packages into **Setup.exe**, to be extracted at run time, if necessary. This option overrides the run-time locations that are specified for each individual package's Location setting in the Packages view. If the entire installation must be self-contained in **Setup.exe**, select this option. Note that the **Download from the Web** option results in smaller installations and shorter download time; however, the **Extract from Setup.exe** option provides for a completely self-contained installation. • **Download from the Web**—Download all of the packages that are included in your project (if necessary) from the URLs that are specified for each package in the Packages view. This option overrides the run-time locations that are specified for each individual package's Location setting in the Packages view. This option is recommended if your Advanced UI or Suite/Advanced UI installation will be downloaded from the Internet and you want to minimize the installation size and download time. A package will not be downloaded if the correct version is already present on the target system. • **Follow Individual Selections**—Use the locations that are specified for each individual package's the Location setting in the Packages view. To learn more, see Specifying the Run-Time Location for Advanced UI or Suite/Advanced UI Packages at the Release Level.
Cache Packages Locally	Advanced UI, Suite/Advanced UI	Specify whether you want to cache the Advanced UI or Suite/Advanced UI packages that are run on a target system in the location that is defined for each package in the Packages view. The default value is Yes. If you are building an uncompressed release, you may want to select No for this setting.

Table 12-129 • Settings on the Setup.exe Tab (cont.)

Setting	Project Type	Description
Update URL	Advanced UI, Suite/Advanced UI	If you may want to make updates for your Advanced UI or Suite/Advanced UI setup launcher available for download when they are available, specify the absolute path, including the file name, for an updated Advanced UI or Suite/Advanced UI setup launcher. This support makes it possible for end users to easily obtain a newer version of your product before installing an earlier version. For more information about updates for your Advanced UI or Suite/Advanced UI installation, including requirements for the update, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.
Expiration Date	Basic MSI, InstallScript MSI	To prevent end users from being able to run **Setup.exe** on or after a certain date, enter the expiration date, or click the arrow in this setting to select the date from a calendar. To remove the expiration date, click the Delete button in this setting. If you specify an expiration date in this setting, use the Expiration Date setting to specify the message that you want to be displayed at run time if an end user tries to launch **Setup.exe** on or after the expiration date.
Expiration Message	Basic MSI, InstallScript MSI	Enter the message that you want to be displayed at run time if an end user tries to launch **Setup.exe** on or after the expiration date that is configured in the Expiration Date setting.

Signing Tab for a Release



Project • *The Signing tab is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

The Signing tab is where you specify the digital signature information—including the digital certificate files that a certification authority grated to you—that InstallShield should use to sign your files. It is also where you specify which files in your installation should be digitally signed at build time.

Table 12-130 • Settings on the Signing Tab

Setting	Project Type	Description
Sign Setup.exe File	Advanced UI, Suite/Advanced UI	Specify whether you want to sign the Advanced UI or Suite/Advanced UI installation.
Certificate URL	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	Type a fully qualified URL—for example, **http://www.mydomain.com**. This URL is used in your digital signature to link to a site that you would like end users to visit to learn more about your product, organization, or company.
Digital Certificate Information	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	To specify the digital certificate that you want to use to sign your release, click the ellipsis button (...) in this setting. The Certificate Selection dialog box opens, enabling you to specify either the location of the .pfx file or information about the certificate store that contains the certificate. To learn more, see Certificate Selection Dialog Box.
Certificate Password	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	If the .pfx file that you are using has a password, enter it. InstallShield encrypts the password and stores it in your project file (.ism). At build time, InstallShield uses the password to sign files with a .pfx file. If your certificate is protected by a password but you do not enter it in this setting, signing with a .pfx file fails. Note that if you configure your project to use a certificate that was imported with password protection into a store, Windows prompts for the password at build time when InstallShield is attempting to sign your project's files. The strong key protection that Windows uses does not permit InstallShield to provide the password to the cryptographic provider.

Table 12-130 • Settings on the Signing Tab (cont.)

Setting	Project Type	Description
Sign Output Files	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify which files you want to be signed. Available options are: • **Media Header**—To sign only your media header file (**Data1.hdr**), select this option. This option is available for InstallScript projects. • **None**—To avoid signing your installation, select this option. This option is available for Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module projects. • **Setup.exe**—To sign your **Setup.exe** file, select this option. This option is available for Basic MSI, InstallScript, and InstallScript MSI projects. • **Setup.exe and Media Header**—To sign your **Setup.exe** file and your media header file (**Data1.hdr**), select this option. This option is available for InstallScript projects. • **Setup.exe and Windows Installer Package**—To sign your **Setup.exe** file and your Windows Installer package (.msi), select this option. This option is available for Basic MSI and InstallScript MSI projects. • **Windows Installer Package**—To sign your Windows Installer package (.msi or .msm), select this option. This option is available for Basic MSI, InstallScript MSI, and Merge Module projects.
Signature Description	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	Specify the signature description that you want to use for files that are specified in the Sign Output Files setting. The description that you specify is displayed on the User Account Control (UAC) box to the right of the "Program Name:" label. The UAC dialog box opens when an end user launches the signed file and elevated privileges are required. If you leave this setting blank, InstallShield uses the name of the file without its extension as the description to the right of the "Program Name:" label on the UAC dialog box. Note that if you use the Sign Files in Package setting and its subsettings to sign the files in your package, InstallShield does not use this signature description for the UAC dialog box of the files in your package that are signed at build time.

Table 12-130 • Settings on the Signing Tab (cont.)

Setting	Project Type	Description
Sign Files in Package	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify whether you want to sign any of the files in your release. If you select Yes, use the Include Patterns and Files setting and the Exclude Patterns and Files setting to indicate which files should be signed.  ***Windows Logo*** • *All executable files (including .exe, .dll, .ocx, .sys, .cpl, .drv, and .scr files) in an installation must be digitally signed for the Windows logo program.*
Sign Files That Are Already Signed	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If any of the files in your project are already digitally signed, determine whether you want InstallShield to replace those existing digital signatures with the digital signature that you specify on the Signing tab. Note that this affects only files that meet the requirements that are specified in the Include Patterns and Files setting and the Exclude Patterns and Files setting. • To use the digital signature information that you are providing on the Signing tab to sign a file instead of any existing digital signature information that is already included with the file, select Yes. • To leave the existing digital signature information intact for any files that are already signed, select No. The default value is No.
Sign Files in Their Original Location	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Determine whether you want InstallShield to sign your original files or just the files that are built into the release: • If you want InstallShield to sign a temporary copy of each file and then use that signed temporary copy to build a release, select No. Note that if you select No, InstallShield will not modify or sign your original files. • If you want InstallShield to sign your original files, select Yes. The default value is No.

Table 12-130 • Settings on the Signing Tab (cont.)

Setting	Project Type	Description
Include Patterns and Files	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To specify the files and file patterns that you want to be digitally signed at build time, do one of the following: • To select one or more file names or file patterns from a list of all of the static files that are currently in your project, as well as file patterns such as *.dll, click the ellipsis button (...) in this setting. The Browse for file dialog box opens, enabling you to select one or more patterns and files. When you are done selecting items, InstallShield adds one or more new Include settings under the Include Patterns and Files setting. • To type a file name or pattern manually, click the Add button in this setting. InstallShield adds a new Include setting under the Include Patterns and Files setting; use this new setting to specify the file name or pattern.
Include	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the file or file pattern that you want to be digitally signed at build time. Note the following guidelines: • To indicate a wild-card character, use an asterisk (*). For example, if you want to sign all .exe files, specify the following: ***.exe** Using wild-card characters is especially helpful if you include dynamically linked files in your project and you want to sign all files that match a certain pattern. • Note that the files and file patterns that should not be signed override any files and file patterns that should be signed. For example, if you specify ***.exe** in an Include setting and in an Exclude setting, InstallShield does not sign any .exe files. To delete the file or file pattern, click the Delete button in this setting. To add another file or file pattern, use the Include Patterns and Files setting.

Table 12-130 • Settings on the Signing Tab (cont.)

Setting	Project Type	Description
Exclude Patterns and Files	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	To specify the files and file patterns that you do not want to be digitally signed at build time, do one of the following: • To select one or more file names or file patterns from a list of all of the static files that are currently in your project, as well as file patterns such as *.dll, click the ellipsis button (...) in this setting. The Browse for file dialog box opens, enabling you to select one or more patterns and files. When you are done selecting items, InstallShield adds one or more new Exclude settings under the Exclude Patterns and Files setting. • To type a file name or pattern manually, click the Add button in this setting. InstallShield adds a new Exclude setting under the Exclude Patterns and Files setting; use this new setting to specify the file name or pattern.
Exclude	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify the file or file pattern that you do not want to be digitally signed at build time. Note the following guidelines: • To indicate a wild-card character, use an asterisk (*). For example, if you do not want to sign any .drv files, specify the following: ***.drv** Using wild-card characters is especially helpful if you include dynamically linked files in your project and you want to avoid signing all files that match a certain pattern. • Note that the files and file patterns that should not be signed override any files and file patterns that should be signed. For example, if you specify ***.exe** in an Include setting and in an Exclude setting, InstallShield does not sign any .exe files. To delete the file or file pattern, click the Delete button in this setting. To add another file or file pattern, use the Exclude Patterns and Files setting.

.NET/J# Tab for a Release



Project • *The .NET/J# tab is available in the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The .NET/J# tab is where you add support for the 32-bit versions of the .NET Framework 1.0, 1.1, or 2.0. It is also where you add J# support to your project.

Note • *To include other versions of the .NET Framework redistributables in your project, use the Redistributables view to add the appropriate InstallShield prerequisite for the Microsoft .NET Framework to your project.*

For more information, see Adding .NET Framework Redistributables to Projects.

Table 12-131 • Settings on the .NET/J# Tab

Setting	Project Type	Description
.NET Framework Location	Basic MSI, InstallScript MSI	Specify where the .NET Framework runtime should be located. The .NET Framework is required for applications that are using any .NET features. Valid options are: • **Copy From Source Media**—Leave the .NET Framework runtime on the root of the source media. This option is not applicable to Web media types or Network Image media types with all of your files compressed into **Setup.exe**. • **Extract From Setup.exe**—Compress the .NET Framework runtime into **Setup.exe**, to be extracted at run time. • **Download from the Web**—Download the .NET Framework runtime (if necessary) from a URL that you specify. If you select this option, you must set the value of the .NET and J# Framework URL setting to the appropriate Web location. • **Do Not Include**—Do not include one of the 32-bit versions of the .NET Framework 1.0, 1.1, or 2.0 in the selected release.
.NET Framework Version	Basic MSI, InstallScript MSI	Select the 32-bit version of the .NET Framework that you want to distribute with your installation. ***Note*** • *To include .NET Framework 3.5, 3.0 SP1, 3.0, 2.0 SP1 (x86, x64, IA64), or 2.0 (only x64, IA64) redistributables in your project, use the Redistributables view to add the appropriate InstallShield prerequisite for the Microsoft .NET Framework to your project.* *For more information, see Adding .NET Framework Redistributables to Projects.*
.NET 1.1/2.0 Core Language	Basic MSI, InstallScript MSI	If you selected .NET version 1.1 for the .NET Framework Version setting, you can specify one .NET core language that you want to distribute. This is the language that is used while the .NET 1.1 core redistributable is installed. If you selected version 2.0, the language options are all selected and disabled since they are all included with this version of the redistributable. To change the core language, click the ellipsis button (...) in this setting.

Table 12-131 • Settings on the .NET/J# Tab (cont.)

Setting	Project Type	Description
Command Line to Pass to Dotnetfx.exe	Basic MSI, InstallScript MSI	***Note*** • *This setting applies only if .NET 1.1 is selected for the .NET Framework Version setting.* Enter the command line that you want to pass to Microsoft's **DotNetFx.exe**.
.NET 1.1/2.0 Language Packs	Basic MSI, InstallScript MSI	To include .NET language packs, click the ellipsis button (...) in this setting. The options that are available depend on the Microsoft language packs that you have installed on your build machine. If you click the ellipsis button, you can download additional available language packs.
Command Line to Pass to Language Packs	Basic MSI, InstallScript MSI	***Note*** • *This setting applies only if .NET 1.1 is selected for the .NET Framework Version setting.* Enter the command line that you want to send to all of the Microsoft **LangPack.exe** files included with the installation.
Display .NET Option Dialog	Basic MSI, InstallScript MSI	Indicates whether **Setup.exe** displays a Yes/No message box asking the end user if they want to install .NET Framework on the target system. This setting does not determine whether your installation includes the .NET Framework, only whether the end user has a choice to install it. • **No**—Installs the .NET Framework, if required, without displaying a message box to the end user. • **Yes**—Displays the .NET option message box, which allows the end user to specify whether to install the .NET Framework. If you select Yes, a message box is displayed on the target system at run time. The message box states that the installation optionally uses the Microsoft .NET Framework and asks if they would like to install it.
Show Full User Interface when Installing .NET Framework	Basic MSI, InstallScript MSI	Specify whether you want the full interface for the .NET Framework installation to be displayed. If you select Yes, the Microsoft .NET Framework (English) Setup wizard appears when **dotnetfx.exe** installs the .NET Framework on the target system. This wizard shows the progress of the .NET Framework installation. If you select No, the InstallShield Wizard appears when **dotnetfx.exe** installs the .NET Framework on the target system. This InstallShield Wizard shows the progress of the .NET Framework installation.

Table 12-131 • Settings on the .NET/J# Tab (cont.)

Setting	Project Type	Description
.NET Build Configuration	Basic MSI, InstallScript MSI	The .NET Build Configuration setting contains the name of the build configuration of the .NET solution. InstallShield uses this setting to determine the location of a project outputs' files (for example, Debug or Release). The C# and VB.NET project wizards automatically update the value of this setting when you create a new project or change the solution for the project.
.NET and J# Framework URL	Basic MSI, InstallScript MSI	Specify the location from which your installation downloads the .NET Framework runtime and the J# redistributable, if included. It is not necessary to specify the file name in the URL. This setting is required only if the **.NET Framework Location** or **J# Redistributable Location** setting (if included) for the current release is set to **Download from the Web**. The installation downloads and runs the InstallShield file **Dotnetfx.exe**, which in turn downloads **Dotnetredist.exe** from the Microsoft Web site. This behavior is hard-coded in **Dotnetfx.exe**, which can be hosted anywhere.
J# Redistributable Location	Basic MSI, InstallScript MSI	Specify where the J# redistributable should be located. The .NET Framework is required for applications that use .NET features. Available options are: • **Copy from Source Media**—Leave the J# redistributable on the root of the source media. This option is not applicable to Web media types or Network Image media types with all of your files compressed into **Setup.exe**. • **Extract from Setup.exe**—Compress the J# redistributable into **Setup.exe**, to be extracted at run time. • **Download from the Web**—Download the J# redistributable (if necessary) from the URL specified in the .NET and J# Framework URL setting. • **Do Not Include**—Do not include the J# redistributable in the selected release.  ***Note*** • *The J# version number that is used matches whatever version number that you select for the .NET Framework. You cannot install version 1.1 of one of these redistributables and version 2.0 of the other.*
Command Line to Pass to the J# Redistributable	Basic MSI, InstallScript MSI	Specify a command-line parameter to pass to **vjredist.exe**. Consult Microsoft Support for valid command-line parameters.

Table 12-131 • Settings on the .NET/J# Tab (cont.)

Setting	Project Type	Description
Display J# Option Dialog	Basic MSI, InstallScript MSI	Specify whether the installation should display a dialog that lets end users indicate if they want to install J# on the target system. ***Note*** • *If the .NET Framework 1.1 is also included in the installation, the dialog states that .NET Framework 1.1 will also be installed.*
Install J# if Installation Runs Silently	Basic MSI, InstallScript MSI	Specify whether you want to install J# on the target system if the J# Option dialog cannot be shown (for example, if the installation is run silently).

Internet Tab for a Release

Project • *The Internet tab is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The Internet tab is where you specify Web-related information.

Table 12-132 • Settings on the Internet Tab

Setting	Project Type	Description
Web Type	Basic MSI, InstallScript MSI	Select the configuration of your Web installation package. Available options are: • **One Executable**—Build this release as a single self-extracting **Setup.exe** file. This Web type is ideal for a package that is to be downloaded from many Web or FTP sites, since the installation package is self-contained. Note, however, that the entire installation package is downloaded, so the download time is longer than with the Install from the Web type that is described below. • **Downloader**—Build this release as a combination of **Setup.exe** and your .msi package, where the end user downloads and launches **Setup.exe**, which in turn downloads and runs the .msi database, which contains all of your files. • **Install from the Web**—Build this release as a combination of **Setup.exe**, your .msi database, and external cabinet files (.cab). The end user downloads and run **Setup.exe**, which in turn downloads and runs the .msi database; based on the end user's setup type and feature selections, only the requested .cab files are downloaded and installed, which minimizes download time. For the Downloader and Install from the Web types, application maintenance and repair uses the URL that you specify in the URL for Your Files setting as the installation source.
URL for Your Files	Basic MSI, InstallScript MSI	Specify the directory that contains the data cabinet files that will be downloaded. This setting accepts a URL in the form **http://www.yourcompany.com/download**. This setting is used only if the Web Type setting for the current release is set to Install from the Web or Downloader.
IFTW .cab Size (KB)	Basic MSI, InstallScript MSI	Specify the size, in kilobytes, of the cabinet files (.cab) for the Web media type. Specify the value 0 (zero) to build a separate .cab file for each component. This setting is used only if the value of the Web Type setting for the current release is Install from the Web.

Table 12-132 • Settings on the Internet Tab (cont.)

Setting	Project Type	Description
Generate One-Click Install	Basic MSI, InstallScript, InstallScript MSI	Specify whether to create a One-Click Install, which is an installation program whose initial user interface is an HTML page. When an end user visits your Web page and clicks an Install button on it, the installation files are downloaded to the target system and then launched. ***Project*** *• For Basic MSI and InstallScript MSI projects: What files are downloaded to the end user's system depends on the option that is selected for the Web Type setting for the current release. If you select Yes for the Generate One-Click Install setting, you must specify file names in the One-Click HTML Base Name setting and the One-Click .cab Base Name setting for this release. This setting is used only if you select Web for the release's Media Format setting.* *For InstallScript projects: If you select Yes for this setting, InstallShield includes with your installation an external setup player (***Setup.ocx** *file) that downloads and then launches the* **Setup.exe** *file with the appropriate command line. For details, see One-Click Install Installations in InstallScript Projects.*
One-Click HTML Base Name	Basic MSI, InstallScript MSI	Specify the base file name of the HTML file that InstallShield should generate at build time. The .htm file name extension is appended to the base name that you specify, and the generated file is created in the Disk1 folder of the current release location. This setting is used only if the value of the Generate One-Click Install setting is Yes.
One-Click .cab Base Name	Basic MSI, InstallScript MSI	Specify the base file name for the cabinet file (.cab) that is generated by the build process for a One-Click Install installation. The .cab file name extension is appended to the name that you specify here. The generated files are created in the Disk1 folder for the current release. The name that you specify for this setting is displayed at run time in the Digital Certificate when you digitally sign your installation. This setting is used only if the value of the Generate One-Click Install setting is Yes.
Create Default Web Page	InstallScript	Specify whether you want InstallShield to create a default Web page (.htm file) and place it in the Disk1 folder.

Table 12-132 • Settings on the Internet Tab (cont.)

Setting	Project Type	Description
Web Page URL	InstallScript	Do one of the following to indicate what URL should be launched if you click the Run From Web command on the Build menu: • To launch the **Setup.htm** file that InstallShield creates at build time and places in the Disk1 folder, leave this box blank. • To launch a different Web page, type the URL in this box or click the browse button to select the Web file.  ***Tip*** • *If you enter a URL, do not include the name of the page, and do not include an ending forward slash. For example, if the full URL for the* **Setup.htm** *file is* ***http://www.mypages.com/setup/Setup.htm****, enter the following as the Web page URL:* **http://www.mypages.com/setup** *When you click the Run from Web command on the Build menu, InstallShield launches the appropriate URL (****http://www.mypages.com/setup/Setup.htm****, in the aforementioned example).*

Events Tab for a Release



Project • *The Events tab is available in the following project types:*

- *Advanced UI*
- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *Suite/Advanced UI*

Additional project-specific differences are noted where applicable.

The Events tab lets you configure settings for various tasks that you want InstallShield to perform before, during, or after builds.

The Events tab settings are organized into the following main categories:

- Commands
- Publishing
- Distribution
- Virtual Machine (This area is available in the Premier edition of InstallShield.)

Commands Settings

Use the Commands area on the Events tab to specify commands that you want to be run at various stages of the build process.

Table 12-133 • Commands Settings on the Events Tab

Setting	Project Type	Description
Prebuild Event	Basic MSI, InstallScript, InstallScript MSI, Merge Module, Suite/Advanced UI	 ***Edition*** • *This setting is available in the Premier edition of InstallShield.* Specify the command that you want to be run before InstallShield starts building the release. This event runs after InstallShield creates the release folder and log file, but before InstallShield starts building the release. To specify more than one command, click the ellipsis button (...) in this setting, which opens the Prebuild Event dialog box. Enter each command on a separate line in this dialog box. At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one. When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands. To learn more, see Specifying Commands that Run Before, During, and After Builds.
Precompression Event	Basic MSI, InstallScript MSI	 ***Edition*** • *This setting is available in the Premier edition of InstallShield.* Specify the command that you want to be run after InstallShield has built the .msi package and the .cab files (if your product's data files are to be stored in .cab files). Note that this event occurs after .cab files are streamed into the .msi package, but before the .msi package has been digitally signed and streamed into the **Setup.exe** file. To specify more than one command, click the ellipsis button (...) in this setting, which opens the Precompression Event dialog box. Enter each command on a separate line in this dialog box. At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one. When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands. To learn more, see Specifying Commands that Run Before, During, and After Builds.

Table 12-133 • Commands Settings on the Events Tab (cont.)

Setting	Project Type	Description
Execute Batch File	InstallScript, InstallScript Object	To launch a batch file (.bat) or executable file (.exe) after the release has been built, enter the path to the file, or click the ellipsis button (...) to browse to the file. You can use path variables, enclosed within angle brackets (<>), as part of the path. If you specify a file for this, InstallShield sets environment variables with the same names (including the angle brackets) and values as the project's build variables. InstallShield also sets the environment variable <ISMEDIADIR>, whose value is the path to the folder in which the release's Disk Images, Log Files, and Report Files folders are created. You can refer to the value of an environment variable in a batch file by surrounding the variable name with percent signs (%); for example: set PATH = %<ISPROJECTDIR>%;%PATH% When the file is finished, InstallShield deletes the environment variables that it set.
Postbuild Event	Basic MSI, InstallScript, InstallScript MSI, Merge Module, Suite/Advanced UI	 ***Edition*** • *This setting is available in the Premier edition of InstallShield.* Specify the command that you want to be run after InstallShield has built and signed the release. To specify more than one command, click the ellipsis button (...) in this setting, which opens the Postbuild Event dialog box. Enter each command on a separate line in this dialog box. At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one. When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands. To learn more, see Specifying Commands that Run Before, During, and After Builds.

Publishing Settings

The Publishing settings that are displayed depend on what project type you are using; they also depend on whether you are using the version of InstallShield that ships by itself, or the one that is included in AdminStudio.

Publishing Settings in the Version of InstallShield that Ships By Itself (Without AdminStudio)

For the version of InstallShield that ships by itself (without AdminStudio), the settings in the Publishing area on the Events tab are available in merge module projects. Use these settings to specify information such as whether you want the merge module to be available in the Merge Modules folder or a repository after each successful build of the release.

Table 12-134 • Publishing Settings on the Events Tab—for InstallShield Without AdminStudio

Setting	Project Type	Description
Publish Merge Module	Merge Module	Specify whether you want the merge module to be available in the Merge Modules folder or a repository after each successful build of the selected release. Available options are: • **Publish to repository**—InstallShield publishes the merge module to the repository that is specified in the Repository Name setting whenever you successfully build the release. Note that this option is available only if one or more repositories have been configured. (To configure a repository, on the Tools menu, click Options, and then click the Repository tab. Use this tab to configure a repository.) • **Copy to Merge Modules folder**—InstallShield updates the Merge Modules folder with this new merge module whenever you successfully build the release. • **Do not publish**—InstallShield does not publish the merge module to a repository or copy it to the Merge Modules folder when you build the release. To learn more, see Using a Repository to Share Project Elements.
Repository Name	Merge Module	Select the repository where the merge module that you are building should be published when the release is successfully built. Note that this option is available only if one or more repositories have been configured. (To configure a repository, on the Tools menu, click Options, and then click the Repository tab. Use this tab to configure a repository.)
Display Name	Merge Module	Specify a display name for the merge module that you are publishing to the repository. When you select this merge module in the Redistributables view after it has been published to the repository, the display name is available in the details pane.
Publisher	Merge Module	Type the name of the person publishing this merge module to the repository. This name will be displayed in the description pane when you select this merge module in the Redistributables view. When you select this merge module in the Redistributables view after it has been published to the repository, the publisher name is available in the details pane.

Table 12-134 • Publishing Settings on the Events Tab—for InstallShield Without AdminStudio (cont.)

Setting	Project Type	Description
Description	Merge Module	Type the description that should be displayed for this merge module. When you select this merge module in the Redistributables view after it has been published to the repository, the description is available in the details pane.

Publishing Settings in the Version of InstallShield that Ships as Part of AdminStudio

For the version of InstallShield that ships as part of AdminStudio, the settings in the Publishing area on the Events tab are available in Basic MSI projects. Use these settings to specify information such as whether you want the Windows Installer package that you are building to be published to the software repository.

Table 12-135 • Publishing Settings on the Events Tab—for InstallShield with AdminStudio

Setting	Project Type	Description
Publish to AdminStudio Repository	Basic MSI	Specify whether you want the Windows Installer package that you are building to be published to the software repository. If you use the software repository to manage a Windows Installer package, the .msi file and all of its other associated files and subfolders are stored in a subfolder of the software repository location that is identified for the package's Application Catalog.
Group	Basic MSI	If you select Yes for the Publish to AdminStudio Repository setting and you want to associate the Windows Installer package with one or more groups in the Application Catalog, click the ellipsis button (...) in this setting. The Select Application Manager Groups dialog box opens. Select the check box for each group that you want to contain the Windows Installer package.
Update Option	Basic MSI	Specify how you want to handle the importing of the Windows Installer package that is built for the selected release into the Application Catalog. Available options are: • **New Package Version**—A new build is treated as a new package in the Application Catalog. • **Overwrite Existing Version**—A new build overwrites the existing version in the Application Catalog. • **New Package History Version**—A new build is treated as a new version of the package that exists in the Application Catalog. • **Ignore if Exists**—A new build is imported only if it does not already exist in the Application Catalog.

Distribution

Use the Distribution area on the Events tab to configure settings for distributing releases to a folder or FTP site automatically at build time.

Table 12-136 • Distribution Settings on the Events Tab

Setting	Project Type	Description
Copy to Folder	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want to be able to automatically distribute your release to a folder, specify location. Existing folders with the same names as copied folders are overwritten, but no folders are deleted. Specify the folder path by entering the path in this setting, or click the ellipsis button (...) to browse to the location. If the media format of the selected release is a network image, which creates only one disk image folder, the contents of the disk image folder, rather than the folder itself, are copied. If you chose to create a self-extracting executable file, the executable file, rather than the disk image folders, is copied.  ***Project*** • *For InstallScript and InstallScript Object projects, InstallShield automatically copies the release to the folder that you specify on the Events tab every time that you build the release.* *For any of the following project types, InstallShield copies all of the relevant files for your release to the specified folder whenever you right-click the release in the Releases view and then click Distribute:* • *Basic MSI* • *InstallScript MSI* • *Merge Module* *If you want the build engine to copy your release to the specified folder after every build in a Basic MSI, InstallScript MSI, or Merge Module project, select Yes for the Distribute After Build setting.*  ***Note*** • *If you specify a folder for this setting and you also specify an FTP location on the Events tab, InstallShield copies the release to only the FTP location.*

Table 12-136 • Distribution Settings on the Events Tab (cont.)

Setting	Project Type	Description
FTP Location	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want to be able to automatically distribute your release to an FTP server, specify the FTP URL for the location. Each release in your project can have a different FTP location. ***Note*** • *If you need to distribute your release to a path outside the FTP default folder, use a double slash (//). For example, to distribute your release to a root-level folder called* ***myproduct****, where the URL of the FTP server is* ***ftp://ftp.mydomain.com****, enter* **ftp://ftp.mydomain.com//myproduct** *for the FTP location.* ***Project*** • *For InstallScript and InstallScript Object projects, InstallShield automatically copies the release to the FTP location that you specify on the Events tab every time that you build the release.* *For any of the following project types, InstallShield copies all of the relevant files for your release to the specified FTP location whenever you right-click the release in the Releases view and then click Distribute:* • *Basic MSI* • *InstallScript MSI* • *Merge Module* *If you want the build engine to copy your release to the specified location after every build in a Basic MSI, InstallScript MSI, or Merge Module project, select Yes for the Distribute After Build setting.*
FTP Site User Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If a user name is required to upload to the FTP location that you are specifying, enter the user name. ***Project*** • *If you enter a user name for one of the releases in a Basic MSI, InstallScript MSI, or Merge Module project, that same user name is used for other releases in the same project and in other Basic MSI, InstallScript MSI, and Merge Module projects.*
FTP Site Password	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If a password is required to upload to the FTP location that you are specifying, enter the password. ***Project*** • *If you enter a password for one of the releases in a Basic MSI, InstallScript MSI, or Merge Module project, that same password is used for other releases in the same project and in other Basic MSI, InstallScript MSI, and Merge Module projects.*

Table 12-136 • Distribution Settings on the Events Tab (cont.)

Setting	Project Type	Description
Distribute After Build	Basic MSI, InstallScript MSI, Merge Module	Specify whether you want the build engine to automatically distribute your release after each build to the location that you specified. ***Tip*** • *To quickly distribute your release on demand, instead of after each build, right-click the release and click Distribute. Note that the Distribute command is available only if the release has been built at least once.*

Virtual Machine Settings

Edition • *The Virtual Machine area is available in the Premier edition of InstallShield.*

Use the Virtual Machine area on the Events tab to configure settings for distributing the selected release to a virtual machine (VM) at build time or on demand. To learn more, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-137 • Virtual Machine Settings on the Events Tab

Setting	Project Type	Description
Configuration	Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	This setting shows the name of the group of VM configuration settings that you want to use when distributing the selected release to a VM. To specify an existing VM configuration, select it in this setting. To create a new VM configuration, click the ellipsis button (...) in this setting. To edit an existing VM configuration, select it in this setting and then click the ellipsis button (...). To override the selected VM configuration's details (such as the destination on the VM where you want the installation to be distributed) for the selected release, configure the subsettings under this setting. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand. For information on how to share VM configurations with other machines, see Sharing Virtual Machine Settings for Distribution of Releases.

Table 12-137 • Virtual Machine Settings on the Events Tab (cont.)

Setting	Project Type	Description
Snapshot	Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	Optionally specify the name of the snapshot that you want to use for distributing the release. If a snapshot is specified for the VM configuration that is selected in the Configuration setting, the name of that snapshot is considered to be the default value for this setting and displayed in gray text. To override the snapshot for the selected release, enter it in this setting. If no snapshot is specified for the VM configuration, InstallShield does not revert to a specific snapshot. InstallShield powers on the VM without reverting it to a specific snapshot, and copies your installation to the VM. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.
User Name	Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	If a user name is specified for the VM configuration that is selected in the Configuration setting, that user name is considered to be the default value for this setting and displayed in gray text. To override the user name for the selected release, enter it in this setting. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.
Password	Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	If a password is specified for the VM configuration that is selected in the Configuration setting, that password is considered to be the default value for this setting and displayed in gray text. To override the password for the selected release, enter it in this setting. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.
Destination Path	Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	The destination path identifies the location on the VM where the release is to be distributed. The last subfolder in the path can be a path that InstallShield creates on the VM; the other folders in the path must already exist. If a destination path is specified for the VM configuration that is selected in the Configuration setting, that destination path is considered to be the default value for this setting and displayed in gray text. To override the destination path for the selected release, enter it in this setting. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-137 • Virtual Machine Settings on the Events Tab (cont.)

Setting	Project Type	Description
Distribute After Build	Basic MSI, InstallScript, InstallScript MSI, Suite/Advanced UI	Specify whether you want InstallShield to automatically distribute the selected release each time that it is successfully built. Distributing the release to the VM involves the following tasks: **1.** Revert the VM to the designated snapshot (if one is specified). **2.** Power on the VM. **3.** Copy the built release to the VM to make it available for testing. As an alternative to automatically distributing the release at build time to the VM, you can distribute the release on demand: right-click the release and then click Distribute to VM. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Windows App Tab for a Release



Project • *The Windows App tab is available in Basic MSI projects.*

UWP app package (.appx) creation is enabled in the per-release tab titled Windows App. Here, some core settings can be specified that impact the UWP app package build process.

The Windows App tab settings are organized into the following main categories:

- General
- Package Identity Overrides
- Display Properties

General Settings

Use the General area on the Windows App tab to specify whether a UWP app package (.appx) will be built as part of building the release and to specify other core settings that impact the build process such as choosing whether to include desktop extensions or server extensions.

Table 12-138 • General Settings on the Windows App Tab

Setting	Project Type	Description

Build UWP App Package	Basic MSI	Specify whether you want a UWP app package (.appx) to be built as part of building this release. Available options are:
		• **Yes**—Build the UWP app package. Choose **Yes** if you want to:
		  ◦ Distribute your app through the Windows Store.
		  ◦ Target Windows Server Nano.
		  ◦ Sideload a Windows App that leverages the Universal Windows Platform (UWP).
		• **No**—Do not build the UWP app package. If you select **No**, only the .msi will be built.

Distribution Method	Basic MSI	Choose a distribution method for your UWP app package. This setting directs which set of warnings are issued for potential problems related to Windows Store policies or requirements related to sideloading. Available drop-down options are: • **Direct Distribution (Sideload)**—Distribute a UWP app package without having it downloaded from the Windows Store. Sideloaded apps have not been certified by the Windows Store. A common example of a sideloaded app is one that is distributed to an enterprise environment and is internal to a company only. If a package will be sideloaded, you will typically need to sign it, unless your customer wishes to sign it themselves. • **Windows Store**—Distribute a UWP app package through the Windows Store. When this option is selected, additional warnings will be issued for items that are known to be rejected by Windows Store policy despite the build being valid in a sideloading scenario. If a package will be distributed through the Windows Store, it is subject to additional policies that limit its contents. This type of app doesn't need to be signed before it is uploaded to Microsoft because Microsoft signs the package.

Table 12-138 • General Settings on the Windows App Tab (cont.)

Setting	Project Type	Description
Include Desktop Extensions	Basic MSI	Choose whether to include "full trust" apps and other Desktop Bridge extensions as part of the core UWP app package (.appx) format. If you want to enable your desktop apps to start leveraging Universal Windows Platform (UWP) APIs, you should enable desktop extensions. If you select **No**, warnings are issued for any such items that cannot be represented. If you select **Yes**, additional subsettings can be configured as needed to specify a minimum Windows version and maximum tested Windows version supported by the included apps and desktop extensions.  ***Tip*** • *When using InstallShield to create a UWP app package, choose to include either desktop extensions or server extensions, but not both. Although a UWP app package can contain both desktop and server extensions, targeting one set over the other is a more likely scenario.*

Minimum Version
 	
(under Include Desktop Extensions)
	Basic MSI	Optionally, specify the app's minimum supported Windows version number. If you leave this blank, the minimum version number of Windows 10 Anniversary Update is assumed.
		
		The version must contain only numbers and it requires four parts using the following format (with values in each part ranging from 0 to 65535): *majorversion.minorversion.buildnumber.revisionnumber*, such as:
		
		`1.0.0.60325`
		
		Use as few characters as possible. For example, do not use unnecessary leading zeros; instead of entering `14.00.25.1`, enter the following:
		
		`14.0.25.1`

Tested Version
 	
(under Include Desktop Extensions)
	Basic MSI	Optionally, specify the app's maximum supported tested Windows version number. If you leave this blank, this setting defaults to match the Minimum Version (or the minimum version number of Windows 10 Anniversary Update if nothing is specified for Minimum Version).
		
		The version must contain only numbers and it requires four parts using the following format (with values in each part ranging from 0 to 65535): *majorversion.minorversion.buildnumber.revisionnumber*, such as:
		
		`1.0.0.60325`
		
		Use as few characters as possible. For example, do not use unnecessary leading zeros; instead of entering `14.00.25.1`, enter the following:
		
		`14.0.25.1`

Table 12-138 • General Settings on the Windows App Tab (cont.)

Setting	Project Type	Description
Include Server Extensions	Basic MSI	Choose whether to include Windows Server extensions as part of the core UWP app package (.appx) format, thus creating a Windows Server App. If you want to target Windows Server Nano's native installation package format and deploy a service, WMI provider, event source, or performance counter, you should enable server extensions. If you select **No**, warnings are issued for any items that cannot be represented. If you select **Yes**, additional subsettings can be configured as needed to specify the minimum Windows Server operating system and maximum tested Windows Server operating system supported by the included apps.  ***Tip*** • *When using InstallShield to create a UWP app package, choose to include either desktop extensions or server extensions, but not both. Although a UWP app package can contain both desktop and server extensions, targeting one set over the other is a more likely scenario.*

Minimum Version
 	
(under Include Server Extensions)
	Basic MSI	Optionally, specify the version number of the app's minimum supported Windows Server operating system. If you leave this blank, the minimum version of Windows Server 2016 is assumed.
		
		The version must contain only numbers and it requires four parts using the following format (with values in each part ranging from 0 to 65535): *majorversion.minorversion.buildnumber.revisionnumber*, such as:
		
		`1.0.0.60325`
		
		Use as few characters as possible. For example, do not use unnecessary leading zeros; instead of entering `14.00.25.1,` enter the following:
		
		`14.0.25.1`

Tested Version
 	
(under Include Server Extensions)
	Basic MSI	Optionally, specify the version number of the app's maximum supported tested Windows server operating system. If you leave this blank, this setting defaults to match the Minimum Version setting (or the minimum version of Windows Server 2016 if nothing is specified for Minimum Version).
		
		The version must contain only numbers and it requires four parts using the following format (with values in each part ranging from 0 to 65535): *majorversion.minorversion.buildnumber.revisionnumber*, such as:
		
		`1.0.0.60325`
		
		Use as few characters as possible. For example, do not use unnecessary leading zeros; instead of entering `14.00.25.1`, enter the following:
		
		`14.0.25.1`

Table 12-138 • General Settings on the Windows App Tab (cont.)

Setting	Project Type	Description
Framework Dependencies	Basic MSI	Select the retail or debug build of the C++ runtime that matches the one used in your product.  ***Note*** • *This setting does not cause the dependency packages to be installed in sideloading scenarios. For those scenarios, the dependency may have to be delivered manually. Also, the debug versions of the of the C++ runtime are only useful for internal testing and cannot be uploaded to the Windows Store. For local testing or sideloading, you may also have to deploy the framework package that is included in the Windows 10 SDK.*
Merge Dependencies	Basic MSI	 ***Edition*** • *This setting is available in the Premier edition of InstallShield.* Specify a list of UWP app packages to merge into the resulting package. Click the ellipsis button (...) to browse for a UWP app package (.appx), or type one or more paths directly in this setting, separated by semicolons. Registry data and files from the VFS are copied into this package, but any duplicates of data already in the package are omitted.

Package Identity Overrides Settings

Use the Package Identity Overrides area on the Windows App tab to uniquely identify a package to Windows. These fields are typically not shown to an end user. Also, because the settings are overrides, the fields are not expanded by default.



Note • *Architecture is also part of a UWP app package's identity. Use the Template Summary setting in the Product Configuration, or in General Information, to specify the architecture of your UWP app package.*

Table 12-139 • Package Identity Overrides Settings on the Windows App Tab

Setting	Project Type	Description
Name	Basic MSI	Specify a unique alphanumeric string that identifies your UWP app package to Windows. If this is left blank, InstallShield generates a name based on the **Product Name** setting in the Product Configuration or General Information.

Table 12-139 • Package Identity Overrides Settings on the Windows App Tab

Setting	Project Type	Description
Publisher	Basic MSI	To override the actual value of the certificate if one is specified on the Signing tab, specify the Subject name (name of the owner) of the certificate that will be used to sign the package. For example: `CN=Flexera Software LLC, O=Flexera Software LLC, L=Itasca, S=Illinois, C=US` If a package will be distributed through the Windows Store, it doesn't need to be signed before it's uploaded to Microsoft, in which case a signing certificate may be skipped. If a package will be sideloaded, it may or may not need to be signed, depending on how your customers expect to handle the package: • If the customer prefers to sign the package themselves, they may need a specific publisher defined in the **Publisher** override setting that matches their certificate. • If the customer merely wants to deploy the package, they may expect it to be signed with a trusted Authenticode Certificate. The requirements on this are equivalent to those for a trusted UAC prompt. • Sideloading requires that the Sideloading setting in Windows is enabled. Sideloading is enabled by default in the Windows 10 Anniversary Update.  ***Tip*** • *The following settings on the Signing tab of the Releases view also influence the built package by controlling whether and how the UWP app package is signed:* ***Digital Certificate Information***, ***Certificate Password***, *and* ***Sign Output Files***.

Version	Basic MSI	Enter a version number for the UWP app package to which the certificate conforms. If nothing is specified, the version number is copied from the **Product Version** setting in General Information or Product Configuration.
		The version must contain only numbers and requires four parts using the following format (with values in each part ranging from 0 to 65535): *majorversion.minorversion.buildnumber.revisionnumber*, such as:
		`1.0.0.60325`
		Use as few characters as possible. For example, do not use unnecessary leading zeros; instead of entering `14.00.25.1`, enter the following:
		`14.0.25.1`

Display Properties

Use the four fields in the Display Properties area on the Windows App tab to identify a package to an end user..

Table 12-140 • Display Properties Settings on the Windows App Tab

Setting	Project Type	Description
Display Name	Basic MSI	Specify the string that users will recognize as the name of the package. This is typically the name of the application. If nothing is specified, the Display Name is copied from the **Product Name** setting in General Information or Product Configuration. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. ***Note*** • *Localization of UWP app packages requires the Windows 10 SDK be installed on the same machine as InstallShield. If it is not present, InstallShield will build a UWP app package containing only the default language.*
Publisher Display Name	Basic MSI	Specify the string that users will recognize as the publisher's name. For example: `Flexera Software LLC` If nothing is specified, the Display Name is copied from the **Publisher** setting in General Information or Product Configuration. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield. ***Note*** • *Localization of UWP app packages requires the Windows 10 SDK be installed on the same machine as InstallShield. If it is not present, InstallShield will build a UWP app package containing only the default language.*

Table 12-140 • Display Properties Settings on the Windows App Tab (cont.)

Setting	Project Type	Description
Description	Basic MSI	Optionally, specify a description to help distinguish the package to an end user. When you type a value for this setting, you are creating a string entry and setting its initial value for all of the languages that are currently in the project. As an alternative to typing a new value, you can click the ellipsis button (...) in this setting to select an existing string. For more information, see Using String Entries in InstallShield.  ***Note*** • *Localization of UWP app packages requires the Windows 10 SDK be installed on the same machine as InstallShield. If it is not present, InstallShield will build a UWP app package containing only the default language.*
Logo	Basic MSI	Specify the image used to represent the package. This is typically a 50x50 logo. If no logo is specified, it is created from the **Display Icon** setting in General Information.

Chained .msi Package Settings



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Windows Installer 4.5 includes support for installing multiple packages using transaction processing. The packages are chained together and processed as a single transaction. If one or more of the packages in the transaction cannot be installed successfully or if the end user cancels the installation, the Windows Installer initiates rollback for all packages to restore the system to its earlier state.

The Chained .msi Packages area in the Releases view is where you identify .msi packages that you want InstallShield to include in your installation as chained packages. This area is also where you configure settings such as the properties that should be passed to the chained packages.

The Windows Installer chaining functionality supports any packages that can be installed through **MsiInstallProduct**; it does not support the chaining of InstallScript MSI packages.

Windows Installer 4.0 and earlier versions do not install any chained .msi packages.



Tip • *To learn how to add Windows Installer 4.5 redistributables, see Adding Windows Installer Redistributables to Projects.*

The following table describes the settings for chained .msi packages.

Table 12-141 • Chained .msi Package Settings

Setting	Description
Installation (run-time path)	Specify the name of the Windows Installer package (.msi file), or click the Browse button to browse to it. If you click the Browse button, InstallShield automatically enters the product code for you. InstallShield also prompts you to specify whether you want to stream the .msi package—and its uncompressed files, if the .msi package is not compressed—into your product's main .msi package: • If you specify that you want InstallShield to stream the files, InstallShield adds the name of the .msi package to the **Installation (run-time path)** setting. InstallShield also automatically adds either the file name (if it is a compressed .msi package) or the entry ***.*** (if it is an uncompressed .msi package) to the **Streamed files** box. For the *.* entry, InstallShield streams in the .msi package, all of the files in the same folder as the .msi package, and all of the subfolders and their files. The value that you use for this setting is relative to a temporary extraction path. That is, you could add a streamed supporting file as **Support\File.ext**, and refer to it in the chained package as **[SourceDir]Support\File.ext**. • If you specify that you do not want InstallShield to stream the files, InstallShield adds the following path to the **Installation (run-time path)** setting: **[SourceDir]FileName.msi** You can change this path if appropriate. For example, you may want to use a path such as the following one, and also clear the **Delete streamed files after installation** check box: **[LocalAppDataFolder]{ProductCodeGUID}\FileName.msi** In this example, the .msi package is cached on the local system, and it is available for maintenance.  ***Tip*** • *You may not want to stream many or large files in the .msi package, since the Windows Installer has limitations for the file size of .msi packages.*
Product code	This setting indicates the product code of the .msi package that is being chained to your main installation. The product code is used for uninstalling the chained package when appropriate. InstallShield automatically adds the product code for this setting if you use the Browse button for the **Installation (run-time path)** setting to select the .msi package.

Table 12-141 • Chained .msi Package Settings (cont.)

Setting	Description
UI level	Specify the user interface (UI) level that you want to be used for the chained .msi package. Valid options are: • **Basic UI (/qb)**—Display the built-in small progress dialog. • **Full UI (/qf)**—Display the modal and modeless dialogs that are available in the .msi package. • **No UI (/qn)**—Run the installation silently. • **Reduced UI (/qr)**—Display only modeless dialogs, such as the full-size progress dialog.
Install condition	Enter any installation condition that you want to use for the chained .msi package. If the condition evaluates to True at run time, the Windows Installer launches the chained .msi package if the chained package's product has not already been installed. The default condition is: **Not Installed** You can modify this if necessary. For example, if you are adding the chained .msi package to a minor upgrade and you want the Windows Installer to launch the chained .msi package during a first-time installation and during an upgrade, consider adding a condition such as the following one: **Not Installed OR REINSTALL><"FeatureName"** In this example, **FeatureName** is the name of a feature that will be updated if the main installation is running in upgrade mode. It will also be updated during a repair.  ***Note*** • *At run time, the installation evaluates the install condition; if it evaluates to False, the installation evaluates the removal condition. If the install condition always evaluates to True (even during uninstallation of the main product), the removal condition is never evaluated, and the chained .msi package is never uninstalled. Therefore, you may want to avoid specifying an install condition that will always evaluate to True.* For information on condition syntax, see Conditional Statement Syntax in the Windows Installer Help Library.

Table 12-141 • Chained .msi Package Settings (cont.)

Setting	Description
Install properties	To pass one or more properties from the main installation to this chained .msi package during the installation, enter the properties and their corresponding values. For example, to install a specific feature of a chained package to a subfolder of the main installation's INSTALLDIR location, you could enter the following string for this setting: **INSTALLDIR="[INSTALLDIR]Subfolder\" ADDLOCAL=Feature1**
Removal condition	Enter any uninstallation condition that you want to use for the chained .msi package. If the Install condition evaluates to False and the Removal condition evaluates to True, Windows Installer removes the chained package's product if it is present. The default condition is: **REMOVE="ALL"** For information on condition syntax, see Conditional Statement Syntax in the Windows Installer Help Library.
Removal properties	To pass one or more properties from the main installation to this chained .msi package during the uninstallation, enter the properties and their corresponding values. This setting is often left blank.
Release flags	You can assign one or more release flags to a chained .msi package that you want to include in only certain builds. For example, if you have a chained .msi package that should be included only in a special edition of your product that contains a special add-on that requires the chained .msi package, you can flag that chained package and include it only when it is needed. If you want to assign a release flag to this chained .msi package, enter it for this setting. The string that you enter can be any combination of letters or numbers. To have more than one flag on a chained package, use a comma to separate the flags.
Streamed files	This box indicates the files for the chained .msi package that are being streamed into the main installation's package. Use the Add File and Add Folder buttons to add more files or entire folders to the list of streamed files. To specify an alternate extraction location, edit the entry's **Run-time path** value by selecting and then clicking it again. InstallShield highlights the entry's text, enabling you to edit it. The files are extracted to a temporary folder by default. To remove an entry in this box, select it and then click the Remove button.

Table 12-141 • Chained .msi Package Settings (cont.)

Setting	Description
Delete streamed files after installation	If you want the installation to delete the streamed files after installation, select this check box. If the streamed files should remain on the target system, clear this check box. It is generally most useful to let the files remain on the system only if you have extracted them to a non-temporary location, such as **[LocalAppDataFolder]{PackageCode}*.***, to cache an extracted package locally. This check box is selected by default for chained .msi packages that included one or more streamed files. This check box is disabled for chained .msi packages that do not include any streamed files.

Patch Design View



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Once you have added an upgrade item in the Upgrades view, you can use the Patch Design view to create a patch configuration for that upgrade so that you can deliver it as a patch. The patch can be applied to target systems that have an earlier version of the product.

You can create multiple patch configurations in the Patch Design view. Each patch configuration contains the settings and data required to build a patch. Each patch configuration must contain at least one latest setup and one previous setup. With these minimal settings, you can create a patch.

Patch Configuration



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

To create a patch, you begin by right-clicking the Patch Design explorer in the Patch Design view and then clicking Add New Patch Configuration. A patch configuration consists of one latest setup and one or more previous setups. Before you specify the latest and previous setups, you can specify overall patch properties on the following tabs for your patch configuration:

- Common
- Identification
- Digital Signature

- Sequence
- Advanced

Common Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you click a patch configuration in the Patch Design view, InstallShield displays several tabs. The Common tab for a patch configuration exposes frequently used patch settings.

Patch Output Location

Specify where you want your patch file built or browse for an existing folder. InstallShield appends the following subdirectories to your output location:

*****PatchConfigName***\Patch**



Note • *InstallShield also adds a folder called Interim to your output directory. The Interim folder contains the following files after you build a patch:*

- *.log—contains output from the patch creation process; this usually contains information that is useful for troubleshooting*
- *.pcp—patch creation properties file; this contains all of the settings necessary to generate a patch package*

Launcher Settings

In this area, you can configure the following launcher settings:

Table 12-142 • Configurable Launcher Settings

Setting	Description
Create Update.exe	If you want to create an **Update.exe** update launcher for the current patch, select this check box. To learn when an **Update.exe** update launcher is required, see Patching Considerations.
Include Windows Installer 3.1 Engine	Select this check box to include the Windows Installer 3.1 engine with your patch package.
Include .NET Framework	Select this check box to include the .NET Framework with your patch package.

Patch Creation Cache

In this area, you can set the following:

Table 12-143 • Patch Creation Cache Settings

Setting	Description
Enable	When you select this check box, InstallShield creates intermediate files that are used in subsequent builds to improve performance speed. These intermediate files are left indefinitely on your system. Do not select this check box if you have limited disk space.

Optimize Patch for Large Files

Select this check box to create small bit-level patches for all application files larger than 4 MB in your installation project.

Allow Patch to be Uninstalled (Requires Windows Installer 3.0)

Select this check box if you would like the patch to be uninstallable without having to uninstall and reinstall the entire application and other patches. Note that uninstallation of patches works only under certain conditions. For example, versions of Windows Installer earlier than version 3.0 cannot remove just the patch from an application. For more information, see Removing Patches in the Windows Installer help.



Project • *The uninstallable patch functionality is not available for InstallScript MSI projects if the InstallScript user interface (UI) style is the traditional style (which uses the InstallScript engine as an external UI handler).*

This functionality is applicable to InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

Build Patch

Click this button to build your patch after you have configured all of the appropriate settings.



Note • *You can also build your patch from the Patch Design explorer. Right-click the appropriate patch configuration, and then click Build Patch.*

Identification Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you click a patch configuration in the Patch Design view, InstallShield displays several tabs. The Identification tab exposes settings for display strings. The display strings are used to populate information about the patch in Add or Remove Programs. It is also used by the Windows Installer 3.0 (and later) APIs that interrogate and catalog patches applied to a target machine.

Tip • *Every time that you change the latest setup for a patch configuration in the Patch Design view, InstallShield uses the Add or Remove Programs information from the latest setup as the values for the Identification tab settings. You can override the values on the Identification tab as needed.*

Table 12-144 • Identification Tab Settings

Setting	Description
Display name	Specify the name of your patch.
Support URL	Specify the uniform resource locator (URL) that you would like your customers to visit for technical support.
Description	Specify a brief description of your patch.
Manufacturer name	Specify the name of the application's manufacturer.
Target product name	Specify the name of the application or the target application suite.
Classification	Specify the category of upgrade. Examples include Critical Update, Hotfix, and Service Pack.

Digital Signature Tab

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you click a patch configuration in the Patch Design view, InstallShield displays several tabs. The Digital Signature tab is where you specify settings if you want to digitally sign your patch.

Note • *If you want to digitally sign the files—such as your application's executable files—in your patch, you can specify which files should be signed through the Signing tab in the Releases view.*

Table 12-145 • Digital Signature Tab Settings

Setting	Description
Sign the Patch Package	To digitally sign your patch package, select this check box.
Sign Update.exe	To digitally sign the **Update.exe** file, select this check box.

Table 12-145 • Digital Signature Tab Settings (cont.)

Setting	Description
Certificate URL	Type a fully qualified URL—for example, **http://www.mydomain.com**. This URL is used in your digital signature to link to a site that you would like end users to visit to learn more about your product, organization, or company.
Digital Certificate Information	To specify the digital certificate that you want to use to sign your release, click the Browse button next to this setting. The Certificate Selection dialog box opens, enabling you to specify either the location of the .pfx file or information about the certificate store that contains the certificate. To learn more, see Certificate Selection Dialog Box.
Password	If the .pfx file that you are using has a password, enter it. InstallShield encrypts the password and stores it in your project file (.ism). At build time, InstallShield uses the password to sign files with a .pfx file. If your certificate is protected by a password but you do not enter it in this setting, signing with a .pfx file fails. Note that if you configure your project to use a certificate that was imported with password protection into a store, Windows prompts for the password at build time when InstallShield is attempting to sign your project's files. The strong key protection that Windows uses does not permit InstallShield to provide the password to the cryptographic provider.
Signature Description	Specify the signature description that you want to use for the patch package and **Update.exe** file, if applicable. The description that you specify is displayed on the User Account Control (UAC) box to the right of the "Program Name:" label. The UAC dialog box opens when an end user launches the signed file and elevated privileges are required. If you leave this setting blank, InstallShield uses the name of the file without its extension as the description to the right of the "Program Name:" label on the UAC dialog box.

Sequence Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you click a patch configuration in the Patch Design view, InstallShield displays several tabs. The Sequence tab is where you specify the order that Windows Installer version 3.0 and later should apply patches to an installed product, regardless of the order in which they are provided to the target machine. For versions of Windows Installer earlier than version 3.0, the patch sequence is ignored, and any patches are applied to the product in the order that they are provided to the target machine.

Use default patch sequencing

Select this check box if you want to use the default sequencing for the patches of your product. This default sequence accounts for obsolete patches, superseded patches, and patches that have already been applied to the product.

If you clear this check box, you can add your own custom sequence. If you clear this check box but do not add a sequence, no sequencing information is built into your patch.

Advanced Tab



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you click a patch configuration in the Patch Design view, InstallShield displays several tabs. The Advanced tab for a patch configuration exposes a comprehensive set of build settings that you can configure for a patch.

Patch Settings

In this area, you can configure the following settings for your patch:

Table 12-146 • Patch Configuration Settings

Property	Description
Patch Output Location	Specify where you want your patch file built or browse for an existing folder. InstallShield appends the following subdirectories to your output location: ***PatchConfigName*\Patch.**  ***Note*** • *InstallShield also adds a folder called Interim to your output directory. The Interim folder contains the following files after you build a patch:* • *.log—contains output from the patch creation process; this usually contains information that is useful for troubleshooting* • *.pcp—patch creation properties file; this contains all of the settings necessary to generate a patch package*
Generate Patch GUID	Specify whether you want a new GUID generated each time that you build your patch. If you select No, the GUID listed for the Patch GUID property is used. The default setting is Yes.
Patch GUID	The patch GUID is used to uniquely identify a patch package. InstallShield automatically generates a new GUID for this property every time you build the patch if the Generate Patch GUID property is set to Yes.

Table 12-146 • Patch Configuration Settings (cont.)

Property	Description
Minimum Windows Installer Version	Specify the minimum version of Windows Installer that is required by the patch. For example, for a minimum of Windows Installer 2.0, enter **200**. For a minimum of Windows Installer 3.0, enter **300**. For a minimum of Windows Installer 3.1, enter **301**. ***Note*** • *If you specify an early version of Windows Installer, you will not be able to use any of the features that are supported by only the latest version of the Windows Installer engine.*

Build Settings

In this area, you can configure the following build settings for your patch:

Table 12-147 • Build Settings for a Patch

Property	Description
Include Whole Files	Specify whether to include whole files for every file included in the patch package. ***Note*** • *You can override this setting on a file-by-file basis in the Latest Setup settings.*
Validate at Build	Specify whether you want to use the upgrading and patching validation every time you build a patch.
Leave Patch Components Decompressed	Select Yes to leave all .msp file information in a temporary folder. This information can be useful when troubleshooting. If you do not want this information stored, select No.
Enable Patch Creation Caching	Select Yes if you want InstallShield to create intermediate files that are used in subsequent builds to improve performance speed. These intermediate files are left indefinitely on your system. Select No if you have limited disk space. The Patch Cache Folder is where you specify where the intermediate files should be created.
Patch Cache Folder	Specify a directory where the temporary files from patch caching should be stored. These intermediate files are left indefinitely on your system.

Table 12-147 • Build Settings for a Patch (cont.)

Property	Description
Generate MsiPatchOldAssembly tables	Specify whether to automatically generate entries for the **MsiPatchOldAssemblyFile** and **MsiPatchOldAssemblyName** tables, which allow a patch package that is running under Windows Installer 3.0 or later to patch an assembly in the global assembly cache (GAC) without making a run-time request for the original installation source. For more information, see Patching Assemblies in the Global Assembly Cache.

Run-time Settings

In this area, you can configure the following run-time settings for your patch:

Table 12-148 • Run-time Settings for a Patch

Property	Description
Allow Product Codes to Differ	Specify whether you want the patch creation executable to suppress build-time prompts if product codes differ between the original and latest installations. For a major upgrade, be sure to select .
Allow Product Versions to Differ	Specify whether you want the patch creation executable to suppress build-time prompts if product versions differ between the original and latest installations.
List of Patch GUIDs to Replace	To replace one or more earlier installed patches with the current patch, set this property to the patch GUIDs of those patches, and separate each with a comma. For example: **{C86838C9-DEDC-4451-B96F-94AFB9460F15},{C8633E5B-AC44-45d8-B487-C68B3B1F60D6}** Setting this property is typically *not* required, even if you have several patch configurations in the Patch Design view. However, if your patch does not overwrite files added in an earlier patch, it may be necessary to set this property.  ***Note*** • *This property affects only the installation portion of the patch. It does not revert files back to their original versions.*
List of Target Product Codes	Specify the product codes that you want to target with this patch, and separate each with a comma. If you enter an asterisk (*) for this setting, InstallShield replaces it at build time by the product codes of the installations that are listed as previous images in this patch configuration.
Create Update.exe	Specify whether you want to create an **Update.exe** update launcher for the current patch. To learn when an **Update.exe** update launcher is required, see Patching Considerations.

Table 12-148 • Run-time Settings for a Patch (cont.)

Property	Description

MSI Command-Line Arguments	Specify Windows Installer arguments to be written to **Setup.ini**.
	`[Startup]`
	`CmdLine=Your Value`
	The default value for this property is
	`REINSTALLMODE=omus REINSTALL=ALL`
	The **REINSTALL** property is a string that contains letters specifying the type of reinstallation to be performed. For more information, see "REINSTALLMODE Property in the Windows Installer Library.

Password Protect Launcher	To password-protect your patch, select Yes, and then type a password for the Launcher Password setting. When you password-protect your patch, any end user who wants to apply your patch must enter a case-sensitive password to launch your update. This setting is applicable only to patches that use an **Update.exe** file.
Launcher Password	Type a password to protect your application. You must select Yes for the Password Protect Launcher setting to activate password protection. When you password-protect your patch, any end user who wants to apply your patch must enter a case-sensitive password to launch your update. This setting is applicable only to patches that use an **Update.exe** file.
Minor Update to Target RTM Version (MSI 3.1 Required)	If you want your minor-upgrade patch to essentially remove all of the patches up to the release to manufacturing (RTM) version of the product (or the most recent major upgrade of the product, if one has been installed) before applying the current minor-upgrade patch, select Yes for this property. With this option, all patches (with or without sequencing data) are removed. You do not need to target additional baseline versions and thus increase the patch payload. All end users can successfully apply the patch without applying any intermediate patches. If you do not want your minor-upgrade patch to remove all of those earlier patches, select No. If you select this option, it may be necessary for your patch to contain the information needed to target each of the earlier minor upgrades that were created after the RTM (or the most recent major upgrade of the product, if one was created). For example, if you are creating a minor-upgrade patch for service pack 2 and you select No for this property, your patch needs to target the minor-upgrade patch for service pack 1. You could also optionally target other baselines (such as RTM); doing so would increase the patch payload. Note that if you do not target the RTM version, any end user who has the RTM version but not the service pack 1 minor-upgrade patch would need to install service pack 1 before service pack 2. Versions of Windows Installer earlier than 3.1 ignore this setting.

Table 12-148 • Run-time Settings for a Patch (cont.)

Property	Description
Optimized Install Mode (MSI 3.1 Required)	If you want Windows Installer 3.1 to optimize the patching process so that it reduces the time that is required to apply the current patch, select Yes. If the patch only modifies a select list of tables, Windows Installer 3.1 ignores the actions that are associated with all other tables. For more information, including the list of tables, see" Patch Optimization" in the Windows Installer Help Library. If you select Yes for this property, Microsoft Corporation recommends that you test your patch extensively to ensure that it behaves as expected. If you do not want Windows Installer 3.1 to optimize the patching process for the current patch, select No. Windows Installer will run a complete repair of the application when the patch is applied to the target machine. Versions of Windows Installer earlier than 3.1 ignore this property. Windows Installer 3.0 optimizes patches if the patch only modifies the select list of tables. To avoid patch optimization by Windows Installer 3.0, you must use the DisableFlyWeightPatching policy. For more information, see DisableFlyWeightPatching in the Windows Installer Help Library. Versions of Windows Installer earlier than 3.0 run a complete repair of the installation when a patch is applied.

Windows Installer Engine

In this area, you can configure the following settings for the Windows Installer engine:

Table 12-149 • Windows Installer Settings for a Patch

Property	Description
Include MSI 3.1 Engine	Specify whether the Windows Installer engine should be included with your patch package.
Engine Location	Select one of the following options: • **Download Engine From The Web**—If the Windows Installer engine needs to be installed, it is downloaded at run time. • **Extract Engine From Update.exe**—The build streams the Windows Installer engine into the **Update.exe** file, giving you a single file to distribute to your customers. At run time, the engine is extracted from the Update.exe file and installed if required. • **Copy From Source Media**—The build copies the Windows Installer engine into the same directory as the **Update.exe** file.

Table 12-149 • Windows Installer Settings for a Patch (cont.)

Property	Description
Windows Installer 3.1 engine URL	Specify the URL for the location of the engine. This is the location that the Update.exe file uses at run time to download the engine. The default URL location is a live site maintained by Flexera Software for your convenience. ***Note*** • *The 3.1 engine is available for download from the default URL (http://www.installengine.com/Msiengine30).*

Microsoft .NET Framework

In this area, you can configure the following settings for the Microsoft .NET Framework:

Table 12-150 • Microsoft .NET Framework Settings for a Patch

Property	Description
Include in Build	Specify whether to include the Microsoft .NET Framework in your patch. If you select Yes, the same version of the .NET Framework that was included in the latest release is included in this new patch.
Engine Location	Select one of the following options: • **Download Engine From The Web**—If the .NET Framework needs to be installed, it is downloaded at run time. • **Extract Engine From Update.exe**—The build streams the .NET Framework into the Update.exe file, giving you a single file to distribute to your customers. At run time, the engine is extracted from the **Update.exe** file and installed if required. • **Copy From Source Media**—The build copies the .NET Framework into the same directory as the **Update.exe** file.
Engine URL	Specify the URL for the location of the .NET Framework. This is the location that the Update.exe file uses at run time to download the .NET Framework. The default URL location is a live site maintained by Flexera Software for your convenience.

Update Launcher Settings

In this area, you can configure the following settings for the **Update.exe** launcher:

Table 12-151 • Update Launcher Settings for a Patch

Property	Description
Company Name	If you want to override the default company name for **Update.exe** with your company name, enter your company name. The company name is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.
Product Name	If you want to override the default product name for **Update.exe** with your product name, enter your product name. The product name is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.
Product Version	If you want to override the default product version for **Update.exe** with your product version, enter your product version. The product version is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.
Description	If you want to override the default description for **Update.exe** with your own description, enter the appropriate description. The description is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.
Copyright	If you want to override the default copyright notice for **Update.exe** with your product's copyright notice, enter your product's copyright notice. The copyright notice is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.

Table 12-151 • Update Launcher Settings for a Patch (cont.)

Property	Description
Required Execution Level	Use the Required Execution Level setting to specify the minimum level that is required by your project's **Update.exe** file for running on Windows Vista and later platforms. The available options are: • **Administrator**—**Update.exe** requires administrative privileges to run. Administrators must authorize it; non-administrators must authenticate as an administrator. • **Highest available**—**Update.exe** prefers administrative privileges. Administrators must authorize it; non-administrators run it without administrative privileges. • **Invoker**—**Update.exe** does not require administrative privileges, and all users can run it without administrative privileges. **Update.exe** does not display any UAC messages prompting for credentials or for consent. • **Use Previous Setup Manifest**—The **Update.exe** manifest uses the same required execution level that was specified for the previous setup. This is the default option. For InstallScript MSI projects, and for Basic MSI projects if the Create Update.exe check box is selected, InstallShield embeds an application manifest in the **Update.exe** launcher. This manifest specifies the selected execution level. Operating systems earlier than Windows Vista ignore the required execution level. If the Create Update.exe check box is cleared for a Basic MSI project, InstallShield does not embed the Windows application manifest in an **Update.exe** launcher. For more information, see Minimizing the Number of User Account Control Prompts During Installation.
Icon	To use your own icon for the **Update.exe** file, specify the fully qualified name of the file that contains the icon. To specify a file, type an absolute path or a path that is relative to a path variable, or click the ellipsis button (...) to browse to the file from within the Change Icon dialog box. By default, the icon with index 0 is used; to specify a different icon, either select an icon in the Change Icon dialog box or append the icon's index or resource ID (preceded by a minus sign) to the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100. If you leave this setting blank, InstallShield uses a default icon for your **Update.exe** file.

Latest Setup



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The Latest Setup item for a patch configuration is where you specify settings for the latest version of your installation.

Latest Setup Path

The latest setup should be the latest version of your installation for which you want to create a patch. The default setting, <ISLatestRelease>, resolves to the last full release that you build.

The release that you specify in this box should be an uncompressed release, ideally created with an administrative installation if the previous product versions were compressed.

Include Whole Files

Select the files that you want to include as whole files in your patch rather than have the build perform a bit-level difference of the files.



Note *• To include all of your files as whole files, set the Include Whole Files property for the patch configuration to Yes.*

Previous Setups



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Each patch configuration in the Patch Design view must contain at least one latest setup and one previous setup in order for you to create a patch. If your patch will target multiple earlier versions of your product, ensure that each target image has its own previous setup item.

For example, if your company supports versions 1.0 and 1.1 of an application, you may want to create a patch that will upgrade both of these applications to version 2.0. To create a single patch to update both versions, you would add V1.0 and V1.1 as previous setups to the latest setup. Your latest setup would contain V2.0.

When you click a previous setup item in the Patch Design view, the following tabs are available:

- Common
- Advanced

Common Tab

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you click a previous setup item in the Patch Design view, InstallShield displays different tabs. The Common tab exposes the basic previous setup configuration settings.

Table 12-152 • Common Settings for Previous Setup Configuration

Option	Description
Previous Setup	Specify the installation (.msi file or Setup.exe file) that needs to be patched to update it to the latest version of your setup. Because the patch-creation process requires an uncompressed release, you will be prompted for a location to decompress the previous version, if necessary. ***Caution*** *• Keep the following in mind when specifying paths. You cannot decompress setups that span multiple disks from a local or network drive. You will need to find an alternate method of decompressing the setup.*
Ignore Missing Files	If any files that are included in two separate versions cannot be found in a more recent version, then the files will be excluded from the patch package unless they have changed in the later version. The advantage of ignoring the files is that only the modified files need to be added to the package. A complete setup image is not required if you choose to ignore missing source files.

Advanced Tab

Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you click a previous setup item in the Patch Design view, InstallShield displays different tabs. The Advanced tab contains the settings that are accessible on the Common tab, in addition to the following transform filter settings:

Table 12-153 • Advanced Settings for Previous Setup Configuration

Option	Description
Version Relationship	Select the condition that describes the required relationship between the latest version of your product and the previous version whose settings you are specifying for the patch. InstallShield will create a transform between the selected previous setup and the latest setup only if the selected condition is met.
Version to Check	Select what part or parts of the three-part version number should be used to determine whether a transform between the selected previous setup and the latest setup should be created, based on the option selected for the Version Relationship. Available options are: • **Check Major, Minor, and Upgrade Versions**—If InstallShield should check all three parts of the previous setup's version number with all three parts of the latest setup's version number, select this option. • **Check Major and Minor Versions**—If InstallShield should check the first two parts of the previous setup's version number with first two parts of the latest setup's version number, select this option. The third part of the version number is ignored. • **Check Major Version Only**—If InstallShield should check the first part of the previous setup's version number with first part of the latest setup's version number, select this option. The second and third parts of the version number are ignored. • **Do Not Check Versions**—If InstallShield should create the transform regardless of the relationship between the previous setup version number and the latest setup version number, select this option. For example, if you select Check Major, Minor, and Upgrade Versions for this property and you select Previous Setup Version <= Latest Setup Version for the Version Relationship property, InstallShield will create a transform for the differences between the previous setup and the latest setup if the previous setup version number is 1.2.3 and the latest setup version number is 1.2.4. However, InstallShield will not create the transform if the previous setup version number is 1.2.0 and the latest setup version number is 1.1.0. For a version number of 1.2.3, 1 represents the major version number, 2 represents the minor version number, and 3 represents the upgrade version number.
Match Product Code	To create a transform only if the previous setup and the latest setup have the same product code, select Yes. If you select No, InstallShield will create a transform between the previous setup and the latest setup, regardless of the product code.
Match Upgrade Code	To create a transform only if the previous setup and the latest setup have the same upgrade code, select Yes. If you select No, InstallShield will create a transform between the previous setup and the latest setup, regardless of the upgrade code.

Table 12-153 • Advanced Settings for Previous Setup Configuration (cont.)

Option	Description
Match Language	To create a transform only if the previous setup and the latest setup have the same language, select Yes. If you select No, InstallShield will create a transform between the previous setup and the latest setup, regardless of the language.

Additional External Files



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Patching an additional external file provides the ability to create a binary file patch against versions of an application file that have not been shipped as part of the previous installation.

For example, if one of your application files was potentially updated by another installation, you should reference that file version so that the file-level patch can be applied successfully at run time.

External File



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

When you click an external file that you added to the External File area in the Patch Design view, InstallShield displays two settings that need to be configured for that file.

For the first box, browse for the file key with which you want to associate the external file, or type the **File** table key for that file.

For the second box, browse for the external file to associate with the **File** key that you specified in the first box.

Additional Tools View



Project • *The Additional Tools view is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

The additional tools provide added functionality for your project. Not all of the items in this view apply to every project type. Click the following links to see the project types for which the items are available.

Dependency Scanners



Project *• The Dependency Scanners view is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The Dependency Scanners view includes three different scanning tools that you can use to identify possible dependencies that you may need to add to your project. These scanners include a Visual Basic project scanner, a static scanner that scans files that are in your project, and a dynamic scanner that scans a running application for any dependencies.

MSI Debugger



Project *• The MSI Debugger view is available in the following project types:*

- *Basic MSI*
- *MSI Database*
- *Transform*

Debug your setup by setting breakpoints at any action or dialog in the User Interface sequence and watching and changing the Windows Installer properties at each step.

Direct Editor



Project *• The Direct Editor is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*

- *Transform*

The Direct Editor is a view in InstallShield that lets you review all of the tables in your project file (.ism) or database file (.msi, .msm, or .mst). In some project types, this view also offers functionality for advanced users who are very familiar with the Windows Installer database format. The Direct Editor also includes a Reference Tracking pane that lets you easily visualize relationships between table records.

Dependency Scanners



Project • *The Dependency Scanners view is available in the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

A file often relies on functions in other files to perform a task. However, you may not be aware of all these other files—known as *dependencies*—when you include your application's files in your installation project. InstallShield offers the following scanning wizards to assist you in identifying and working with these files. You can access these scanners in the Dependency Scanners view.

Table 12-154 • Dependency Scanners

Scanning Option	Description
Perform Static Scanning	The Static Scanning Wizard looks at portable executable files (for example, .exe, .ocx, .com, .tlb, .hlp, and .chm) in your project and checks for dependencies that they may require.
Perform Dynamic Scanning	The Dynamic Scanning Wizard monitors your system while an executable file is running and checks for .dll or .ocx files that may be required by the executable file.

Any files that are added to a Basic MSI or InstallScript MSI project through one of these scanners are added in accordance with Setup Best Practices.

If you use the Static and Dynamic scanning wizards, you can specify files that you want to be included or excluded automatically any time that you perform a static or dynamic scan through InstallShield. For more information, see Filtering Files in Dependency Scanners.

MSI Debugger



Project • *The MSI Debugger view is available in the following project types:*

- *Basic MSI*
- *MSI Database*

- *Transform*

When you debug a release in the MSI Debugger, you can view and set Windows Installer properties as you step through the package's User Interface and Execute sequences.



Task ***Follow the steps below to begin going through a setup in the MSI Debugger:***

1. Build your release. However, there is one important restriction when you intend on debugging the release: You cannot debug a package that is compressed inside Setup.exe.
2. Go to the **MSI Debugger** view. The debugger lists every standard action and custom action in the User Interface and Execute sequences as well as every dialog in your project. You can also click the **Debug** button on the toolbar or choose **Debug** from the **Build** menu to launch the MSI Debugger.
3. Set a breakpoint on an action or dialog.
4. Start the debugger.

The MSI Debugger runs through each action and dialog until it reaches your breakpoint, at which point it halts execution. Now, you can view and set properties in the Watch window and the Variable window. Finally, you can step through each of the remaining actions, or you can stop the debugger.



Note *• The MSI Debugger supports deferred custom actions that delay the execution of a system change to the time when the installation script is executed.*

Do not confuse the MSI Debugger with the InstallScript Debugger, since they have completely separate purposes. You cannot debug a setup package with the InstallScript Debugger, and you cannot debug an InstallScript custom action with the MSI Debugger.



Tip *• When InstallShield steps into one of the following types of custom actions (You can step into a custom action by pressing the F11 key.), it will give you the option to launch a registered debugger installed on a client machine. A dialog will appear allowing you to choose one of two options. Select Yes to launch a registered debugger. Select No to step over to the next action. The supported custom actions are as follows:*

- *MSI DLL*
- *Standard DLL*

Direct Editor



Project *• The Direct Editor is available in the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

Note that the following information includes additional project-specific details.

The Direct Editor is a view in InstallShield that lets you review all of the tables in your project file (.ism) or database file (.msi, .msm, or .mst). In some project types, this view also offers functionality for advanced users who are very familiar with the Windows Installer database format.

For Windows Installer–based projects, the Direct Editor can run in two different modes:

- **Project edit mode**—This mode lets you edit tables in the project file. The changes that are made in the Direct Editor are incorporated into the Windows Installer package that InstallShield creates at build time.

  Any changes that you make in the other views within InstallShield are reflected in the Direct Editor; in addition, any changes that you make in the Direct Editor are also reflected in the other corresponding views (if available). For example, if you use the Features view to add a new feature to your project, InstallShield automatically adds that feature to the **Feature** table in the Direct Editor. If you use the Direct Editor view to add the feature, InstallShield automatically updates the Features view accordingly.

  When you are working in any of the following Windows Installer–based project types, you are working in project edit mode: Basic MSI, DIM, InstallScript MSI, Merge Module, and QuickPatch.

- **Direct edit mode**—This mode lets you edit tables in the Windows Installer database (.msi, .msm, or .mst file). When you save the changes that you have made in the Direct Editor, InstallShield updates the Windows Installer database.

  Many of the other standard InstallShield views are not available in direct edit mode, since they require build-time functionality that is not available when you are directly editing the Windows Installer database.

  When you are working in any of the following Windows Installer–based project types, you are working in direct edit mode: MSI Database, MSM Database, and Transform.

Using the Direct Editor in Any Project Type

The Direct Editor has a Tables explorer that lists each table in your project or database. When you select a table in this explorer, the right pane shows the following elements:

- A row of buttons and other controls
- A spreadsheetlike table

Each row in a table represents a record in your project or database. The parenthetical note in each column heading indicates the type and size of data that you enter in the column. For example, *S255* indicates a string with up to 255 characters; *I2* indicates a 2-byte integer.

The following table describes all of the buttons and other controls that are displayed when you select a table in the Direct Editor.

Table 12-155 • Controls in the Direct Editor View

Name of Control	Icon	Description
New Record		Displays the Add Record to Table dialog box, which lets you adds a new row to the selected table.
Edit Selected Record		Displays the Edit Record in Table dialog box, which lets you edit the data in the selected row.
Delete Selected Records		Deletes the selected row or rows.
Cut Selected Records		Removes the currently selected rows and places them on the Clipboard.
Copy Selected Records		Copies the currently selected rows to the Clipboard.
Paste Records		Inserts the rows that are saved to the Clipboard.
Find String		Displays the Find dialog box, which lets you search for instances of a string. This dialog box lets you specify criteria such as whether you want to search in the selected table, or all tables.
Find Next		Searches for the next occurrence of the specified string.
Find and Replace		Displays the Replace dialog box, which lets you search for instances of a string and replace them with a new string. This dialog box lets you specify criteria such as whether you want to search in the selected table, or all tables.
Show Reference Tracking		Toggles the Reference Tracking pane below the Direct Editor. The Reference Tracking pane lets you easily visualize relationships between table records. For more information, see Table Record Reference Tracking.
Search Grid		Dynamically filters the records that are displayed in the Direct Editor view according to the string that you specify in this search box. As you type a string in this box, InstallShield hides all of the rows that do not contain it.
Direct Editor Help		Displays the help for the Direct Editor view.

Using the Direct Editor with Project Edit Mode in Windows Installer–Based Projects

In Basic MSI, DIM, InstallScript MSI, Merge Module, and QuickPatch projects, advanced users can use the Direct Editor to perform tasks such as the following:

- View all of the tables in the project file (.ism).
- Add and remove records from tables.
- Cut, copy, and paste records or fields.
- Edit individual fields in the tables.
- Add custom tables.
- Filter the table records that are shown to hide ones that do not contain a specific string.
- Search one table or all tables for a specific string and replace it if necessary.
- Resize and reorder the columns in a table.



***Tip** • If you press F1 while a standard Windows Installer table is selected, the Windows Installer Help Library opens to provide information about that specific table.*

Note the following details when using the Direct Editor with project edit mode in Windows Installer–based projects:

- The **File** table displays only static data. InstallShield may add additional information, such as dynamically linked files, to the **File** table in the Windows Installer database that it creates at build time.
- Unlike the corresponding tables in the Windows Installer database, tables such as the **Binary**, **Icon**, and **Patch** tables in the .ism file do not store binary data. Rather, they store links to build-source paths.
- Column attributes for both standard tables and InstallShield tables cannot be altered while in project edit mode. They can, however, be edited for custom tables. Note that column attributes can be edited in standard, InstallShield, and custom tables in direct edit mode.
- You cannot use localizable properties in the **Directory** table.

Using the Direct Editor with Direct Edit Mode in Windows Installer–Based Projects

In MSI Database, MSM Database, and Transform projects, you can use the Direct Editor to essentially perform all of the tasks that are available in this view with direct edit mode. However, instead of working with tables in the .ism file, you are working directly with the Windows Installer database (.msi, .msm, or .mst).

Using the Direct Editor in InstallScript and InstallScript Object Projects

The Direct Editor in InstallScript and InstallScript Object projects lets you see all of the tables in your .ism file; however, it is recommended that you use the other views in InstallShield to modify these types of projects.

Note that although InstallScript and InstallScript Object projects use many of the same tables that are available in Windows Installer–based projects, many of the common tables have different meanings. In addition, InstallShield ignores custom tables in InstallScript and InstallScript Object projects; custom tables are not available at run time.

Direct Editor Table Insights

The Direct Editor in InstallShield includes several features that provide visual insights into tables, schema information, and validation errors designed to boost the productivity of setup authors or packagers who use the Direct Editor for troubleshooting to identify and resolve advanced problems. These features are described following:

- Directory Table Psuedocolumn Displays Resolved Target Directory Path
- Column Header Schema Information Tooltips
- Table Record Reference Tracking
- Broken References Indicator

Directory Table Psuedocolumn Displays Resolved Target Directory Path

InstallShield includes a read-only, grayed out pseudocolumn to the Directory table that displays the resolved path of each row's directory location. A pseudocolumn behaves like a table column, but is not actually stored in the table. You can sort by the pseudocolumn's visible text, but you cannot insert, update, or delete its values.

Column Header Schema Information Tooltips

InstallShield provides tooltips on column headers that display the column's schema information.

Examples

The **Directory** table receives the following schema tooltips:

Column Header	Tooltip
Directory (s72)	Directory: char(72), required
Directory_Parent (S72)	Directory_Parent: char(72), nullable
DefaultDir (l255)	DefaultDir: char(72), localizable, required
ISDescription (S255)	ISDescription: char(255), nullable
ISAttributes (I4)	ISAttributes: long integer, nullable
ISFolderName (S255)	ISFolderName: char(255), nullable

The **Binary** table receives the following schema tooltips:

Column Header	Tooltip
Name (s72)	Name: char(72), required
Data (V0)	Data: stream, nullable
ISBuildSourcePath (S255)	ISBuildSourcePath: char(255), nullable

The **CustomAction** table receives the following schema tooltips:

Column Header	Tooltip
Action (s72)	Action: char(72), required
Type (i2)	Type: short integer, required
Source (S64)	Source: char(64), nullable
Target (S0)	Target: char(0), nullable
ExtendedType (I4)	Extended Type: long integer, nullable
ISComments (S255)	ISComments: char(255), nullable

Table Record Reference Tracking

The Direct Editor includes a Reference Tracking pane that lets you easily visualize relationships between table records. A **Show Reference Tracking** button at the top of the Direct Editor provides a show/hide toggle of the pane.

Each record may reference one or more other records or be referenced by one or more other records. When a record is highlighted that refers to other records or is referred from other records, the Reference Tracking pane is populated with a **Reference Tables** section showing the tables in which the references reside and an additional section that displays the actual record references. The record references section includes arrow icons that indicate the direction of the reference, where:

- A green arrow pointing to the right indicates that the record that is selected in the Direct Editor table **references** the record displayed in the Reference Tracking pane.
- A blue arrow pointing to the left indicates that the record selected in the Direct Editor table is **referenced by** the record displayed in the Reference Tracking pane.
- Two arrows in both directions (a green arrow pointing right and a blue arrow pointing left) indicate that the record selected in the Direct Editor table **references** *and is* **referenced by** the record displayed in the Reference Tracking pane.



***Note** • When multiple Direct Editor records are selected, only the "focused" record's references are shown. In addition, if multiple tables appear in the **Reference Tables** section, this indicates that the record selected in the Direct Editor table references or is referenced by records in multiple tables. You can click any table in the **Reference Tables** section to view the associated records references.*



***Tip** • In the Reference Tracking pane, you can quickly jump between record references by double-clicking within a cell.*

Broken References Indicator

There are instances where a Direct Editor table record might reference a foreign key record that no longer exists. InstallShield displays the cells of such broken references with a red fill color to call attention to the broken reference.

For example, if the **Directory_** column in the **Component** table references a directory name that is not found in the **Directory** table, then the **Directory_** column is filled in red.

Note • *The Direct Editor broken reference indicator is unrelated to the* ***Maintain referential integrity*** *check box on the Preferences tab of the Options dialog box. While the* ***Maintain referential integrity*** *setting is designed to update the foreign keys when you modify a primary key, the purpose of the broken reference indicator is to display broken references to help you easily identify orphaned records. Therefore, broken references are displayed regardless of the selection of the* ***Maintain referential integrity*** *setting.*

Windows Installer Table Reference

For information about Windows Installer tables, see Database Tables in the MSDN Library.

InstallShield Table Reference

The following table lists some InstallShield tables that add specific functionality to your application. Click a link for more information about a specific table.

Table 12-156 • InstallShield Table Names and Descriptions

Table Name	Project Type	Description
ISAlias	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	This table contains alias information used when an InstallShield Professional project is converted to a Windows Installer–based project or an InstallScript project in the latest version of InstallShield.
ISComCatalogAttribute	Basic MSI, DIM, InstallScript MSI, Merge Module	This table contains information about COM+ catalog attributes. Each entry has a foreign key to a **ISComCatalogObject** table entry to which the attribute belongs.
ISComCatalogCollection	Basic MSI, DIM, InstallScript MSI, Merge Module	This table contains information about COM+ catalog collections. A COM+ catalog collection is a folder of a same type of COM+ catalog objects. For example, the COM+ applications item, the Components item, and the Legacy Components item in the Component Services view are the COM+ catalog collections. Each entry has a foreign key to a **ISComCatalogObject** table entry to which the collection belongs.
ISComCatalogCollectionObject	Basic MSI, DIM, InstallScript MSI, Merge Module	This table contains information about COM+ catalog collections where COM+ catalog objects belong.

Table 12-156 • InstallShield Table Names and Descriptions (cont.)

Table Name	Project Type	Description
ISComCatalogObject	Basic MSI, DIM, InstallScript MSI, Merge Module	This table contains information about COM+ catalog objects. A COM+ catalog object is an item that is contained within a COM+ catalog collection. Each entry has the display name.
ISComPlusApplication	Basic MSI, DIM, InstallScript MSI, Merge Module	This table contains information about a COM+ application. A COM+ application is a COM+ catalog object with extra information. This table has application specific information and a foreign key to a **ISComCatalogObject** table entry which has the basic information.
ISCustomActionReference	Basic MSI, DIM, InstallScript MSI, Merge Module	This table contains information about the behavior of each of the custom actions that are included in the installation.
ISIISItem	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	This table contains information about each IIS element (Web site, application, virtual directory, application pool, and Web service extension) that is configured in the Internet Information Services view.
ISIISProperty	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	This table contains values for settings that are configured in the Internet Information Services view.
ISProductConfigurationInstance	Basic MSI	This table provides support for installing multiple instances of a product in the same context on the same machine.
ISSelfReg	Basic MSI, DIM, InstallScript MSI, Merge Module	If you have selected ISSelfReg as the self-registration method for COM servers, and if your project contains any files (or dynamic links) marked as self-registered, InstallShield adds information about those files to the **ISSelfReg** table of your .msi database.

ISAlias Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

If you upgrade a project that was created with InstallShield Professional to InstallShield 2016, InstallShield might add data to the **ISAlias** table.

In Windows Installer, identifiers cannot contain spaces and must contain only specified characters. Because of this, some of your component names, feature names, or script-defined folder names that you used in InstallShield Professional might not be valid in a project that you upgraded to a Windows Installer–based project. To address this issue, InstallShield uses aliasing to match the old name with a new identifier. (InstallShield does not perform a lexicon change for these names because they might be used as strings within your script.)



Note • *If you upgrade to an InstallScript project, the only invalid characters in feature names or script-defined folder names are the following:*

\ / : * ? " < > |

In component names, the preceding characters and the single quotation mark (') are invalid.

The **ISAlias** table contains the following information.

Table 12-157 • ISAlias Table Information

Column Name	Information
Alias	Name string used in InstallShield Professional.
Identifier	String that is used by InstallShield to reference the alias.
Table	Name of the table that uses the alias (for example, Feature). This table is available in the Direct Editor.

ISCOMCatalogAttribute Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The **ISCOMCatalogAttribute** table contains the information about COM+ catalog attributes. Each entry has a foreign key to a **ISComCatalogObject** table entry to which the attribute belongs.

Columns

ISCatalogObject_(primary key)

A foreign key into the **ISCatalogObject** table

ItemName (primary key)

The named attribute for a catalog object

ItemValue

A value associated with the attribute defined in ItemName

ISCOMCatalogCollection Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The **ISCOMCatalogCollection** table contains information about COM+ catalog collections. A COM+ catalog collection is a folder of a same type of COM+ catalog objects. For example, COM+ application, Component and Legacy Component nodes in the Component Services view are the COM+ catalog collections. Each entry has a foreign key to a **ISComCatalogObject** table entry to which the collection belongs.

Columns

ISCatalogCollection (primary key)

A unique key for the **ISCatalogCollection** table.

ISCatalogObject_

A foreign key into the **ISCatalogObject** table. This is the catalog object that a catalog collection belongs to.

Name

A catalog collection name.

ISCOMCatalogCollectionObject Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The **ISCOMCatalogCollectionObject** table contains information about COM+ catalog collections where COM+ catalog objects belong.

Columns

ISCatalogCollection_(primary key)

A unique key for the **ISCatalogCollection** table

ISCatalogObject_(primary key)

A foreign key into the **ISCatalogObject** table; This is the catalog object that a catalog collection contains.

ISCOMCatalogObject Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The **ISCOMCatalogObject** table contains information about COM+ catalog objects. A COM+ catalog object is an item that is contained within a COM+ catalog collection. Each entry has the display name.

Columns

ISCatalogObject (primary key)

A unique key for the **ISCatalogObject** table

DisplayName

The display name of a catalog object

ISCOMPlusApplication Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The **ISCOMPlusApplication** table contains information about a COM+ application. A COM+ application is a COM+ catalog object with extra information. This table has application specific information and a foreign key to an **ISComCatalogObject** table entry that has the basic information.

Columns

ISCatalogObject_(primary key)

A foreign key into the **ISComCatalogObject** table. This is the root level application object. All other data associated with this application can be derived through the **ISComCatalogObject** table.

ComputerName

If the ComponentService object was loaded from a remote machine, store the computer name here. This can be NULL for the local computer.

Component_

A foreign key to the component table.

ISCustomActionReference Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The **ISCustomActionReference** table contains information about the intended behavior of each custom action that is included in the project.

Table 12-158 • ISCustomActionReference Table Information

Column	Type	Key	Nullable	Description
Action_	Identifier	Yes	No	Identifier that references an entry in the **CustomAction** table.
Description	Text	No	No	Text that describes the behavior of the custom action. When InstallShield builds a release of a Basic MSI, InstallScript MSI, or merge module project, it populates this column with the text in the file that is referenced in the ISCAReferenceFilePath column. If you are working on a project in Direct Edit mode, InstallShield streams the contents of the file into this column as soon as you select the help file in the Custom Actions and Sequences view (or the Custom Actions view).
FileType	Text	No	No	The file type of the help file.
ISCAReference FilePath	Text	No	No	The full path, including the file name, for the file that describes the behavior of the custom action. This column is included in the InstallShield project file (.ism), but not in the .msi file.

ISIISItem Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

The **ISIISItem** table contains information about IIS settings that are configured in the Internet Information Services view.

Table 12-159 • ISIISItem Table Information

Column	Type	Key	Nullable	Description
ISIISItem	String	Yes	No	Primary key for table. This item identifies the Web site or other IIS item that is being created.
ISIISItem_Parent	String	No	Yes	Foreign key into the ISIISItem column. This item identifies the Web site or other IIS item that contains the current item.
DisplayName	Localizable string	No	Yes	An optional localized name for the Web site or other IIS item.
Type	Integer	No	No	Identifies the type of IIS item. Available options are: • 1—Web site • 2—Application • 3—Virtual directory • 4—Application pool • 5—Web service extension
Component_	String	No	Yes	Name of the component that contains the IIS item.

ISIISProperty Table



Project • *This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

The **ISIISProperty** table contains information about IIS settings that are configured in the Internet Information Services view.

***Note** • When you configure advanced settings for a Web site, application, or virtual directory in the Internet Information Services view, InstallShield configures the MetaData* fields in the **ISIISProperty** table automatically. For help on specific settings, see IIS Metabase Properties in the MSDN Help Library.*

Table 12-160 • ISIISProperty Table Information

Column	Type	Key	Nullable	Description
ISIISProperty	String	Yes	No	Primary key for table.
ISIISItem	String	Yes	No	Primary key for table; foreign key into the **ISIISItem** table. This item identifies the Web site or other IIS item that this property is for.
Schema	String	No	Yes	This column is reserved for future use.
FriendlyName	String	No	Yes	This column is reserved for future use.
MetaDataProp	Integer	No	Yes	Equates to the dwMDIdentifier field of the METADATA_RECORD structure.
MetaDataType	Integer	No	Yes	Equates to the dwMDDataType field of the METADATA_RECORD structure. ***Note** • InstallShield supports the following values:* • *1 (DWORD_METADATA)* • *2 (STRING_METADATA)* • *5 (MULTISZ_METADATA)*

Table 12-160 • ISIISProperty Table Information (cont.)

Column	Type	Key	Nullable	Description
MetaDataUserType	Integer	No	Yes	Equates to the dwMDUserType field of the METADATA_RECORD structure. Available options, which are defined in the Windows SDK header **IISCnfg.h**, are: • 1 (IIS_MD_UT_SERVER) • 2 (IIS_MD_UT_FILE) • 100 (IIS_MD_UT_WAM) • 101 (ASP_MD_UT_APP)
MetaDataAttributes	Integer	No	Yes	Equates to the pbMDAttributes field of the METADATA_RECORD structure.
MetaDataValue	String	No	Yes	Equates to the pbMDData field of the METADATA_RECORD structure.
Order	Integer	No	Yes	The order that you would like the property to be applied to the IIS item.
ISAttributes	Integer	No	Yes	Indicates whether the IIS property is a standard property or an advanced property. Advanced properties are configurable through the Other IIS Properties setting in the Internet Information Services view. Available options are: • 0 (Standard property) • 1 (Advanced property) All other options are reserved for future use.

ISProductConfigurationInstance Table



Project • *This information applies to the following project types:*

- *Basic MSI*

The **ISProductConfigurationInstance** table provides support for installing multiple instances of a product in the same context on the same machine. This table defines each of the property values for each instance that an installation supports. For more information, see Setting Properties for an Instance.



Caution • *Creating an installation that lets end users install multiple instances of a product on the same machine and in the same context requires sophisticated authoring and serious commitment on the part of the installation developer. This functionality is recommended for only advanced installation developers.*

Table 12-161 • ISProductConfigurationInstance Table Information

Column	Type	Key	Nullable	Comments
ISProductConfiguration_	Text	Yes	No	Identifies the product configuration to which this instance applies.
InstanceId	Integer	Yes	No	Identifies the instance number of this instance. This value is stored in the property InstanceId.
Property	Identifier	Yes	No	Identifies the property to create for this instance.
Value	Text	Yes	No	Specifies the value for this property for this instance.

ISSelfReg Table



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

If you have selected ISSelfReg as the self-registration method for COM servers, and if your project contains any files (or dynamic links) marked as self-registered, InstallShield adds information about those files to the **ISSelfReg** table of your .msi database.

Table 12-162 • ISSelfReg Table Information

Column Name	Description
FileKey	Foreign key into the File table, identifying the file (.dll, .ocx, .exe, .tlb, .olb) to be self-registered.
Cost	Reserved for future use.
Order	A nonnegative number identifying the order in which self-registration will occur. Any files with an Order value of 1 will be registered first, followed by files with an Order value of 2, and so forth. Finally, all files with an Order value of 0 will be registered.

Table 12-162 • ISSelfReg Table Information (cont.)

Column Name	Description
CmdLine	For self-registering .exe files, stores a command line to be passed to the executable when it is self-registered. By default the field is empty, and the InstallShield run-time engine registers the .exe file with the /regserver argument and unregisters it with the /unregserver argument. To specify different arguments to pass during registration and unregistration, separate the arguments with a vertical bar (|), as in /hello|/goodbye.

InstallShield Columns in Standard Windows Installer Tables

InstallShield extends some standard Windows Installer tables for increased functionality. Following is a listing of the tables affected, columns added, and descriptions of the data to be placed in the new fields.

Table 12-163 • Additions to Standard MSI Tables

Table Name	Column Name	Description
AdminExecuteSequence	ISComments	Author's comments on this Sequence.
AdminUISequence	ISComments	Author's comments on this Sequence.
AdvtExecuteSequence	ISComments	Author's comments on this Sequence.
AdvtUISequence	ISComments	Author's comments on this Sequence.
Binary	ISBuildSourcePath	Full path to the ICO or EXE file.
Component	ISAttributes	This is used to store InstallShield custom properties of a component. Currently the only one is ExtractAtBuild.
Component	ISComments	User Comments.
Control	ISWindowStyle	A 32-bit word that specifies non-MSI window styles to be applied to this control.
Control	ISControlId	A number used to represent the control ID of the Control. Used in Dialog export.
CustomAction	ISComments	Author's comments for this custom action.
Dialog	ISComments	Author's comments for this dialog.
Dialog	ISWindowStyle	A 32-bit word that specifies non-MSI window styles to be applied to this control. This is only used in Script Based Setups.

Table 12-163 • Additions to Standard MSI Tables (cont.)

Table Name	Column Name	Description
Dialog	ISResourceId	A Number that specifies the Dialog ID to be used in Dialog Export.
Directory	ISDescription	Description of the folder.
Feature	ISReleaseFlags	Release Flags that specify whether this feature will be built in a particular release.
Feature	ISComments	Comments.
File	ISBuildSourcePath	Full path, the category is of Text instead of Path because of potential use of path variables.
File	ISAttributes	This field contains the following attributes: • OverrideSystemAttributes (0x04) • OverrideSystemSize (0x08) • OverrideSystemVersion (0x10) • OverrideSystemLanguage (0x20) The UseSystemSettings(0x1) attribute is no longer used.
Icon	ISBuildSourcePath	Full path to the ICO or EXE file.
Icon	ISIconIndex	Optional icon index to be extracted.
InstallExecuteSequence	ISComments	Author's comments on this Sequence.
InstallUISequence	ISComments	Author's comments on this Sequence.
Patch	ISBuildSourcePath	Full path to patch header.
ProgId	ISAttributes	This is used to store InstallShield custom properties of a component, like ExtractIcon, etc.
Property	ISComments	User Comments.
Registry	ISAttributes	This is used to store InstallShield custom properties of a registry item. Currently the only one is Automatic.
Shortcut	ISComments	Author's comments on this shortcut.

QuickPatch Projects

A QuickPatch project is a specific type of Windows Installer–based project recommended for installation authors who want to ship small, single upgrades to their end users. Each QuickPatch project has its own set of views.

Patch Settings View

The Patch Settings view contains links to several views:

- **General Information View**—Enter information about your upgrade and the original installation. Specify which releases should be patched by the current project, and select which custom actions should be included.
- **Files View**—Add files to your patch, and specify file-patching options.
- **Registry View**—Specify which registry values should be changed or removed.
- **Path Variables View**—Make your patch easily portable between development systems with variable-based file linking.

Additional Tools View

The Additional Tools view contains a link to the Direct Editor. Use the Direct Editor to edit the database tables directly.

Patch Settings View



Project • *This information applies to QuickPatch projects.*

The Patch Settings view is available when you open or create a QuickPatch project. Use this view to configure your QuickPatch.

General Information

Enter information about your upgrade and the original installation. Specify which releases should be patched by the current project, and select which custom actions should be included.

Files

Add files to your patch, and specify file-patching options.

Registry

Specify which registry values should be changed or removed.

Path Variables

Make your patch easily portable between development systems with variable-based file linking.

General Information View



Project • *This information applies to QuickPatch projects.*

The General Information view contains basic information about your QuickPatch project. You can view and configure product properties, build settings, patch history, and custom actions in this view.

Product Properties



Project • *This information applies to QuickPatch projects.*

When you click Product Properties in the General Information view, InstallShield displays the following:

Table 12-164 • Settings for Product Properties

Setting	Description
Original Setup Path	This area shows the product name and the path to the original installation image. The path box is read-only.
Product Version	The Original setup version box shows the product version of the original installation. This box is read-only. In the QuickPatch Version box, you can change the product version for your QuickPatch project. Entry in this box is optional.
FlexNet Connect Integration	If FlexNet Connect was enabled in your original installation and if you designate custom version numbers that FlexNet Connect should use to identify your product (as opposed to using the standard version scheme that Windows Installer uses), type the new custom version number for your QuickPatch in this box.

Build Settings



Project • *This information applies to QuickPatch projects.*

When you click Build Settings in the General Information view, InstallShield displays the following tabs:

- Common
- Identification
- Digital Signature
- Advanced

Common Tab



Project • *This information applies to QuickPatch projects.*

When you click Build Settings in the General Information view, InstallShield displays several tabs. The Common tab exposes frequently used build settings for a QuickPatch:

Build Location

Specify where you want your patch file built or browse for an existing folder.

Launcher Settings

In this area, you can configure the following launcher settings:

Table 12-165 • Configurable Launcher Settings

Setting	Description
Create Update.exe	If you want to create an **Update.exe** update launcher for the current QuickPatch package, select this check box. To learn when an **Update.exe** update launcher is required, see Patching Considerations.
Include Windows Installer 3.1 engine	Select this check box to include the Windows Installer 3.1 engine with your patch package.
Include .NET Framework	Select this check box to include the .NET Framework with your patch package.

Patch Uninstallation

Select the **Allow Patch to be Uninstalled (Requires Windows Installer 3.0)** check box if you would like the QuickPatch to be uninstallable without having to uninstall and reinstall the entire application and other QuickPatch packages. Note that uninstallation of QuickPatch packages works only under certain conditions. For example, versions of Windows Installer earlier than version 3.0 cannot remove just the QuickPatch from an application. For more information, see Removing Patches in the Windows Installer help.

Identification Tab



Project • *This information applies to QuickPatch projects.*

When you click Build Settings in the General Information view of a QuickPatch project, InstallShield displays several tabs. The Identification tab exposes settings for display strings. The display strings are used to populate information about the patch in Add or Remove Programs. It is also used by the Windows Installer 3.0 (and later) APIs that interrogate and catalog patches applied to a target machine.



***Note** • Patch metadata is stored directly in the patch (.msp) file and not in any of the .msi packages. The .msp file contents are not localizable. Therefore, string entries are not available for any of the metadata settings below.*

Table 12-166 • Identification Tab Settings

Setting	Description
Display name	Specify the name of your patch.
Support URL	Specify the uniform resource locator (URL) that you would like your customers to visit for technical support.
Description	Specify a brief description of your patch.
Manufacturer name	Specify the name of the application's manufacturer.
Target product name	Specify the name of the application or the target application suite.
Classification	Specify the category of upgrade. Examples include Critical Update, Hotfix, and Service Pack.

Digital Signature Tab



***Project** • This information applies to QuickPatch projects.*

When you click Build Settings in the General Information view of a QuickPatch project, InstallShield displays several tabs. The Digital Signature tab is where you specify settings if you want to digitally sign your patch.



Note** • With QuickPatch projects, you can digitally sign the patch package and the* **Update.exe** *file. If you want to digitally sign individual files—such as your application's executable files—in your QuickPatch package, you must manually sign them and then add them to your project. You can use* **SignTool.exe, which is included with the Windows SDK, to manually sign your files.*

Table 12-167 • Settings in the Digital Signature Tab

Setting	Description
Sign the Patch Package	To digitally sign your QuickPatch package, select this check box.
Sign Update.exe	To digitally sign the **Update.exe** file, select this check box.
Certificate URL	Type a fully qualified URL—for example, **http://www.mydomain.com**. This URL is used in your digital signature to link to a site that you would like end users to visit to learn more about your product, organization, or company.

Table 12-167 • Settings in the Digital Signature Tab (cont.)

Setting	Description
Digital Certificate Information	To specify the digital certificate that you want to use to sign your release, click the Browse button next to this setting. The Certificate Selection dialog box opens, enabling you to specify either the location of the .pfx file or information about the certificate store that contains the certificate. To learn more, see Certificate Selection Dialog Box.
Password	If the .pfx file that you are using has a password, enter it. InstallShield encrypts the password and stores it in your project file (.ism). At build time, InstallShield uses the password to sign files with a .pfx file. If your certificate is protected by a password but you do not enter it in this setting, signing with a .pfx file fails. Note that if you configure your project to use a certificate that was imported with password protection into a store, Windows prompts for the password at build time when InstallShield is attempting to sign your project's files. The strong key protection that Windows uses does not permit InstallShield to provide the password to the cryptographic provider.
Signature Description	Specify the signature description that you want to use for the patch package and **Update.exe** file, if applicable. The description that you specify is displayed on the User Account Control (UAC) box to the right of the "Program Name:" label. The UAC dialog box opens when an end user launches the signed file and elevated privileges are required. If you leave this setting blank, InstallShield uses the name of the file without its extension as the description to the right of the "Program Name:" label on the UAC dialog box.

Advanced Tab



Project • *This information applies to QuickPatch projects.*

When you click Build Settings in the General Information view, InstallShield displays several tabs. The Advanced tab exposes a comprehensive set of build settings that you can configure for a QuickPatch.

Build Location

In the Build Location area, you can configure the following settings.

Table 12-168 • Build Location Settings

Property	Description
Build Location	Specify where you want your patch file built or browse for an existing folder.

Table 12-168 • Build Location Settings (cont.)

Property	Description
Create Update.exe	Specify whether you want to create an **Update.exe** update launcher for the current QuickPatch package. To learn when an **Update.exe** update launcher is required, see Patching Considerations.
List of Patch GUIDs to replace	To replace one or more earlier installed patches with the current QuickPatch, set this property to the patch GUIDs of those patches, and separate each with a comma. For example: **{C86838C9-DEDC-4451-B96F-94AFB9460F15},{C8633E5B-AC44-45d8-B487-C68B3B1F60D6}** Setting this property is typically *not* required, even if you have several QuickPatch projects in the History. However, if your QuickPatch project does not overwrite files added in an earlier QuickPatch project, it may be necessary to set this property. If you do not know the GUID of a patch that you want to replace, click this property and then click the ellipsis button (...). Select the patch (.msp or .exe file), and InstallShield will add the corresponding GUID to this property.
Create new "UpdatedImage" folder	This setting enables you to build your QuickPatch package with an existing UpdatedImage folder. To use this option, you must have built this package at least once (so that an UpdatedImage folder exists). Setting this option to No builds the package from the existing UpdatedImage folder. This lets you tweak the .msi package in the UpdatedImage folder and use that .msi data in a QuickPatch project. Setting this option to Yes (the default setting) regenerates the .msi file in the UpdatedImage folder every time that the package is built.
Generate MsiPatchOldAssembly tables	Specify whether to automatically generate entries for the MsiPatchOldAssemblyFile and MsiPatchOldAssemblyName tables, which allow a patch package that is running under Windows Installer 3.0 and later to patch an assembly in the global assembly cache (GAC) without making a run-time request for the original installation source. For more information, see Patching Assemblies in the Global Assembly Cache.
Create patch sequencing entry	Specify whether you want to use patch sequencing for your QuickPatch. A patch sequence accounts for obsolete patches, superseded patches, and patches that have already been applied to the product. The sequence specifies the order that Windows Installer version 3.0 and later should apply patches to an installed product, regardless of the order in which they are provided to the target machine. For versions of Windows Installer earlier than version 3.0, the patch sequence is ignored, and any patches are applied to the product in the order that they are provided to the target machine.

Table 12-168 • Build Location Settings (cont.)

Property	Description
Streamline QuickPatch	Specify whether you want InstallShield to streamline the creation of your QuickPatch package to build as simple a package as possible. The default value is Yes. The goal of QuickPatch streamlining is to generate a QuickPatch package that has fewer new subfeatures and custom actions than a non-streamlined QuickPatch package. For example, if your QuickPatch project includes a new file or registry entry and InstallShield does not use QuickPatch streamlining, InstallShield creates a new subfeature for that file or registry entry. InstallShield also adds one or more prebuilt InstallShield custom actions to work around certain Windows Installer patch requirements. However, if InstallShield does use QuickPatch streamlining, the file or registry entry is added to an existing feature, and no special prebuilt InstallShield custom actions are required.  ***Note*** • *InstallShield cannot streamline the creation of a QuickPatch package in the following scenarios:* • *The QuickPatch package removes an installed file.* • *The QuickPatch package removes or renames a registry key.* • *The QuickPatch package targets a non-streamlined QuickPatch image. That is, you cannot use QuickPatch streamlining if you select the check box in the History area of the General Information view for a QuickPatch that did not use QuickPatch streamlining. If you try to build a streamlined QuickPatch that targets one or more non-streamlined QuickPatch images, InstallShield displays a build warning, and it does not use streamlining.*
Password Protect Launcher	To password-protect your QuickPatch package, select Yes, and then type a password for the Launcher Password setting. When you password-protect your QuickPatch package, any end user who wants to apply your QuickPatch must enter a case-sensitive password to launch your update. This setting is applicable only to QuickPatch packages that use an **Update.exe** file.
Launcher Password	Type a password to protect your application. You must select Yes for the Password Protect Launcher setting to activate password protection. When you password-protect your patch, any end user who wants to apply your patch must enter a case-sensitive password to launch your update. This setting is applicable only to patches that use an **Update.exe** file.

Windows Installer Engine

In this area, you can configure the following settings for the Windows Installer engine:

Table 12-169 • Windows Installer Engine Settings

Property	Description
Include Windows Installer 3.1 engine	Specify whether the Windows Installer engine should be included with your patch package.
Engine Location	Select one of the following options: • **Download Engine From The Web**—If the Windows Installer engine needs to be installed, it is downloaded at run time. • **Extract Engine From Update.exe**—The build streams the Windows Installer engine into the **Update.exe** file, giving you a single file to distribute to your customers. At run time, the engine is extracted from the Update.exe file and installed if required. • **Copy From Source Media**—The build copies the Windows Installer engine into the same directory as the Update.exe file.
Windows Installer 3.1 engine URL	Specify the URL for the location of the engine. This is the location that the **Update.exe** file uses at run time to download the engine. The default URL location is a live site maintained by Flexera Software for your convenience.

Microsoft .NET Framework

In this area, you can configure the following settings for the Microsoft .NET Framework:

Table 12-170 • Microsoft .NET Framework Settings

Property	Description
Include In Build	Specify whether to include the Microsoft .NET Framework in your patch.
Engine Location	Select one of the following options: • **Download Engine From The Web**—If the .NET Framework needs to be installed, it is downloaded at run time. • **Extract Engine From Update.exe**—The build streams the .NET Framework into the **Update.exe** file, giving you a single file to distribute to your customers. At run time, the engine is extracted from the Update.exe file and installed if required. • **Copy From Source Media**—The build copies the .NET Framework into the same directory as the **Update.exe** file.
Engine URL	Specify the URL for the location of the .NET Framework. This is the location that the **Update.exe** file uses at run time to download the .NET Framework. The default URL location is a live site maintained by Flexera Software for your convenience.

Update Launcher Settings

In this area, you can configure the following settings for the **Update.exe** launcher:

Table 12-171 • Update Launcher Settings for a Patch

Property	Description
Company Name	If you want to override the default company name for **Update.exe** with your company name, enter your company name. The company name is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.
Product Name	If you want to override the default product name for **Update.exe** with your product name, enter your product name. The product name is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.
Product Version	If you want to override the default product version for **Update.exe** with your product version, enter your product version. The product version is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.
Description	If you want to override the default description for **Update.exe** with your own description, enter the appropriate description. The description is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.
Copyright	If you want to override the default copyright notice for **Update.exe** with your product's copyright notice, enter your product's copyright notice. The copyright notice is displayed on the Properties dialog box for the update launcher; this Properties dialog box opens when end users right-click the **Update.exe** file and then click Properties. To learn more, see Customizing File Properties for the Update Launcher.

Table 12-171 • Update Launcher Settings for a Patch (cont.)

Property	Description
Required Execution Level	Use the Required Execution Level setting to specify the minimum level that is required by your project's **Update.exe** file for running on Windows Vista and later platforms. The available options are: • **Administrator**—**Update.exe** requires administrative privileges to run. Administrators must authorize it; non-administrators must authenticate as an administrator. • **Highest Available**—**Update.exe** prefers administrative privileges. Administrators must authorize it; non-administrators run it without administrative privileges. • **Invoker**—**Update.exe** does not require administrative privileges, and all users can run it without administrative privileges. **Update.exe** does not display any UAC messages prompting for credentials or for consent. • **Use Previous Setup Manifest**—The **Update.exe** manifest uses the same required execution level that was specified for the previous setup. This is the default option. For InstallScript MSI projects, and for Basic MSI projects if the Create Update.exe check box is selected, InstallShield embeds an application manifest in the **Update.exe** launcher. This manifest specifies the selected execution level. Operating systems earlier than Windows Vista ignore the required execution level. If the Create Update.exe check box is cleared for a Basic MSI project, InstallShield does not embed the Windows application manifest in an **Update.exe** launcher. For more information, see Minimizing the Number of User Account Control Prompts During Installation.
Icon	To use your own icon for the **Update.exe** file, specify the fully qualified name of the file that contains the icon. To specify a file, type an absolute path or a path that is relative to a path variable, or click the ellipsis button (...) to browse to the file from within the Change Icon dialog box. By default, the icon with index 0 is used; to specify a different icon, either select an icon in the Change Icon dialog box or append the icon's index or resource ID (preceded by a minus sign) to the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100. If you leave this setting blank, InstallShield uses a default icon for your **Update.exe** file.

History



Project • *This information applies to QuickPatch projects.*

The History item in the General Information explorer within the General Information view presents a synopsis of your QuickPatch project. It lists all of the associated releases, enabling you to specify which ones should be patched by the current QuickPatch project.

Custom Action



Project • *This information applies to QuickPatch projects.*

The Custom Action item in the General Information explorer within the General Information view lists the custom actions that are defined in the original installation project for which you are creating a patch. It enables you to specify which actions should be executed by the current QuickPatch project.

Files View



Project • *This information applies to QuickPatch projects.*

The Files view enables you to manage the files in your QuickPatch and also see version numbers, languages, and other information about the files in your original installation. All of the files in the Files To Patch and Original Setup Files explorers are listed by key file.

When you click a file in the Original Setup Files explorer, the file name, the destination, the version number, and other pertinent information are displayed in the right pane. You can drag and drop files from this explorer to the Files To Patch explorer, which shows all of the files that will be added, changed, or removed when your QuickPatch is applied to the original installation.

New File Settings



Project • *This information applies to QuickPatch projects.*

When you click a new file in the Files To Patch explorer within the Files view, you can configure the following settings:

File Settings

Table 12-172 • Settings in the File Settings Area

Setting	Description
Specify the file that you want to add to the patch setup	If you are modifying a file in the original installation, specify the latest version of the file that you want to be installed on the target system when the patch is applied.
Self-Registering	If the file that you are adding is a self-registering DLL or .ocx file, select this check box. ***Note*** • *This option does not self-register .exe or .tlb files.*
Extract COM Information	If the file is a COM server and you want InstallShield to extract COM information from the file when the patch is built, select this check box.

Integration Settings

Select the file destination and the features that you want to associate with this file.

Note • *This install state binding requires feature association. When you add new data to your QuickPatch project, you must associate it with a feature. New data is installed only if the corresponding feature is installed.*

Modified/Deleted File Settings

Project • *This information applies to QuickPatch projects.*

When you click a file that you have added from your original installation to the Files To Patch explorer within the Files view, you can configure the following settings:

Updated File

Table 12-173 • Settings in the Updated File Area

Setting	Description
Specify the latest version of your file	If you are modifying a file in the original installation, specify the latest version of the file that you want to be installed on the target system when the patch is applied.
Include Whole File	If you want to include the selected file in your patch as a whole file, rather than only the byte-level file differences of the file, you may want to consider selecting this check box. If you do so, InstallShield configures your patch so that Windows Installer ignores any actual version number of the selected file, and instead considers it be version 1.0.0.0 when it is determining whether to update the target system's file with the version in your QuickPatch package, or to leave the file as is. Selecting this check box for an unversioned file may be helpful to ensure that the file on the target system is always overwritten with the newer equivalent unversioned file in your QuickPatch package. According to Windows Installer file overwrite rules, a file of any version is maintained over an unversioned file. Therefore, if you select this check box, Windows Installer updates the unversioned file with the one in the QuickPatch package because it considers the file in the QuickPatch to be a versioned file.  ***Important*** • *According to Windows Installer file overwrite rules, the file with the highest version is maintained, even if the file already on the target system has a higher version than the one being installed. Therefore, if the file on the target system is a versioned file, you may want to avoid selecting this check box, since in some scenarios it could lead to the file on the target system not being updated. For example, if the file on the target system is version 1.1.0.0, the file in your QuickPatch project is version 2.0.0.0, and you select this check box, the file on the target system is not updated at run time. This occurs because even though the actual version number of the file in the QuickPatch package is newer than the one on the target system, Windows Installer considers the file in the QuickPatch to be 1.0.0.0, which would make it older than the file on the target system.*
Extract COM Information	If the file is a COM server and you want InstallShield to extract COM information from the file when the patch is built, select this check box. This check box is selected by default for patch files that contain COM information in the **Class** or **TypeLib** tables in the base .msi package.  ***Note*** • *When you specify an existing file that you want to patch, InstallShield automatically detects if the file is self-registering. If the original file was self-registering, the file in the patch is also set to self-register.*

Original File Information

The Original File Information area provides information about the original file in the original installation.

Component Information

The Component Information area provides file component information such as the component name and key file.

Check this box to have the patch delete this file from the setup

Select this check box to remove the file from the target machine when the patch is applied.

Registry View



Project • *This information applies to QuickPatch projects.*

The Registry view in a QuickPatch project provides a visual representation of what currently exists on a source machine and what you will patch on a destination/target system once your patch project is built and applied to the target system. The Destination computer's Registry view pane and the Destination computer's registry data pane are prepopulated with registry entries in the original installation.

Each item in the Destination computer's registry data pane appears next to an icon specific to the value type and the data's current condition. Icons for data values that have been modified have a turquoise pencil. Icons for new data values have a red star in the upper-right corner. Icons for registry values that will be deleted from the target system have a red X. See the following sample icons and descriptions below.

Table 12-174 • Icons in the Registry View of a QuickPatch Project

Icon	Description
	This icon represents a new DWORD value that will be added to the target system when the QuickPatch is applied.
	This icon represents a modified DWORD value that will be updated on the target system when the QuickPatch is applied.
	This icon represents an existing DWORD value on the target system from an earlier installation.
	This icon represents a DWORD value that exists on the target system but will be deleted when the QuickPatch project is applied.

InstallShield Prerequisite Editor Reference

Project *• The following project types include support for InstallShield prerequisites:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

All of these project types include support for the setup prerequisite type of InstallShield prerequisite. Basic MSI projects include support for the feature prerequisite type.

InstallShield also includes support for including InstallShield prerequisites as packages in Advanced UI and Suite/Advanced UI projects. For more information, see Including InstallShield Prerequisites (.prq) in an Advanced UI or Suite/Advanced UI Project.

An InstallShield prerequisite is an installation for a product or technology framework that is required by your product. Some examples of InstallShield prerequisites that are included with InstallShield are Java Runtime Environment (JRE) and SQL Server Express Edition. You can add any of the existing InstallShield prerequisites to your installation projects and configure many of their settings. You can also create your own InstallShield prerequisites, and add them to your projects.

Task ***To open the InstallShield Prerequisite Editor:***

On the **Tools** menu, click **Prerequisite Editor**.

Task ***To open an existing prerequisite in the InstallShield Prerequisite Editor, do one of the following:***

1. On the **File** menu, click **Open**. The **Open** dialog box opens.
2. Select the file by browsing and click **Open**.

The following tabs are associated with the InstallShield Prerequisite Editor:

- Properties
- Conditions
- Files to Include
- Application to Run
- Behavior
- Dependencies

Properties Tab

The Properties tab of the InstallShield Prerequisite Editor is where you specify general information about an InstallShield prerequisite.

Table 12-1 • Settings on the Properties Tab

Setting	Description
Unique identifier for InstallShield prerequisite	Type a unique file identifier for the InstallShield prerequisite or leave the default value as is. This could be the name of the prerequisite application or component, or a GUID. ***Note*** • *Every time you open the InstallShield Prerequisite Editor to create a new InstallShield prerequisite, a new GUID is generated and listed in this box.*
Alternate location to download .prq from if prerequisite files are being downloaded	If appropriate, type the alternate URL for your .prq file. For example: **http://www.mywebsite.com/MyPrq.prq** For more information, see Specifying an Alternate URL for a .prq File. ***Note*** • *InstallShield prerequisites do not have support for FTP URLs.*
Description	Type an overview of the prerequisite. This description is displayed under the Overview heading when you select a prerequisite in the Redistributables view.

Conditions Tab

The Conditions tab of the InstallShield Prerequisite Editor lets you define installation conditions that determine whether an InstallShield prerequisite already is installed on the target machine. Failure to do so causes problems because the target system behaves as if the InstallShield prerequisite was not properly installed. You can also create installation conditions that specify operating system, registry, or file requirements.

If a condition evaluates as true at run time, that condition is met. The InstallShield prerequisite is installed on the target machine if the following are true:

- The target machine meets any of the operating system conditions and all of the other conditions that are listed on the Conditions tab.
- For feature prerequisites only (that is, an InstallShield prerequisite that is associated with one or more features in the main installation)—The feature that contains the feature prerequisite must be installed. Thus, if the feature has a condition that is not met on the target system, or if the end user chooses not to install the feature, the feature is not installed. As a result, none of its associated feature prerequisites are installed, unless the feature prerequisites are also associated with other features that are installed.

Note • *Because installation of prerequisites such as Windows service packs is not supported, the operating system version information is not expected to change. Therefore, the operating system conditions are not considered in verifying whether a prerequisite was installed correctly.*

When you click the Add button on this tab, the Prerequisite Condition dialog box opens. This dialog box also opens when you select an existing condition on the Conditions tab and then click the Modify button.

Files to Include Tab

When you are creating a new InstallShield prerequisite, you must specify the installation files that should be included with the InstallShield prerequisite. You can also modify the list of files for an existing InstallShield prerequisite. The Files to Include tab of the InstallShield Prerequisite Editor is where you specify the files.

When you click the Add button or the Add Multiple Files button on this tab to specify an InstallShield prerequisite file or the Modify button to change the file, the New File dialog box opens.

When you click the **Set File(s) URL** button, the **Set File(s) URL** dialog box opens.

Note • *InstallShield prerequisites do not have support for FTP URLs.*

Application to Run Tab

The Application to Run tab of the InstallShield Prerequisite Editor is where you specify how an InstallShield prerequisite should be installed on the target machine.

Table 12-2 • Settings on the Application to Run Tab

Setting	Description
Specify the application you wish to launch	Select the file—typically a **Setup.exe** setup launcher or an .msi package—that should be launched on the target machine if the InstallShield prerequisite is installed. Only files that have been specified on the Files to Include tab are included in this list. ***Project*** • *If you specify an .msi file and you indicate on the Behavior tab that the progress should be shown, the* **Setup.exe** *setup launcher captures progress messages and uses Windows Installer APIs instead of* **MsiExec.exe** *to launch the .msi package at run time.* If you specify any other file type, or if you specify an .msi file for which progress should not be shown, the **Setup.exe** setup launcher runs the file with either the open verb (for .msi and .exe files) or the default verb (for all other file types) at run time.

Table 12-2 • Settings on the Application to Run Tab (cont.)

Setting	Description
Requires Windows Installer engine and/or .NET Framework to be installed first	If the Windows Installer engine, the .NET Framework (**Dotnetfx.exe**), or both must be installed before this InstallShield prerequisite is installed, select this check box.  ***Note*** • *Selecting this check box does not add the Windows Installer engine or the .NET Framework to your installation. It only specifies that if they are included in the installation, they should be installed before this InstallShield prerequisite is installed. To include the Windows Installer engine or the .NET Framework with your installation, you must add them to your project. To learn how, see Adding Windows Installer Redistributables to Projects or Adding .NET Framework Redistributables to Projects.*
Specify the command line for the application	If applicable, type the command line for the file selected in the **Specify the application you wish to launch** list. Do not include the name of the file in this box. For information on command-line parameters that you can specify, see Specifying Command-Line Parameters for an InstallShield Prerequisite.  ***Note*** • *If a setup prerequisite is configured to be hidden (that is, it is not included in the setup prerequisite run-time dialog as one of the prerequisites that need to be installed), it is launched with its silent command-line parameters—not its standard command-line parameters. Therefore, if it is possible that the **The prerequisite should be hidden from the installation list** check box on the Behavior tab will be selected, specify command-line parameters in the **Specify the command line for the application when the setup is running in silent mode** box.*
Specify the command line for the application when the setup is running in silent mode	If applicable, type the appropriate command line. Do not include the name of the file in this box. For information on command-line parameters that you can specify, see Specifying Command-Line Parameters for an InstallShield Prerequisite.  ***Note*** • *Using the `/s` command-line parameter to launch an installation that includes an InstallShield prerequisite does not automatically run the prerequisite installation silently. You may also need to specify a valid silent command-line parameter for the InstallShield prerequisite in this **Specify the command line for the application when the setup is running in silent mode** setting on the Application to Run tab.*

Table 12-2 • Settings on the Application to Run Tab (cont.)

Setting	Description
Specify the return code (in decimal) the application returns if a reboot is required	If the selected InstallShield prerequisite application requires that the target machine be restarted after the application is installed, type the return code in this box. ***Tip*** • *If multiple return codes exist, list each one separated by a comma.* If you do not know the return codes for the file that you are launching as the InstallShield prerequisite, contact the author of the file. For more information on InstallShield prerequisites that require a restart, see Restarting a Target Machine for InstallShield Prerequisite Installations.

Behavior Tab

The Behavior tab of the InstallShield Prerequisite Editor is where you specify what should occur in certain scenarios.

Table 12-3 • Settings on the Behavior Tab

Setting	Description
The prerequisite requires administrative privileges	Select this check box if administrative privileges are required to install the InstallShield prerequisite or if the InstallShield prerequisite must be installed per machine. This check box is selected by default. ***Note*** • *InstallShield prerequisites that require administrative privileges may require that end users provide credentials or consent before installation can occur on Windows Vista and later systems. For more information, see Minimizing the Number of User Account Control Prompts During Installation.*
The prerequisite may be optionally skipped by the user	If the installation should display a message box that enables end users to choose whether to install the InstallShield prerequisite, select this check box. If end users should not be able to choose whether to install the InstallShield prerequisite, clear this check box.

Table 12-3 • Settings on the Behavior Tab (cont.)

Setting	Description
The prerequisite should be hidden from the installation list	If a target system needs one or more setup prerequisites to be installed, the setup prerequisite dialog is displayed at run time before the main installation runs. When an end user clicks the Install button on this dialog, the setup prerequisites are installed. If you want to include the current setup prerequisite in the list of setup prerequisites that are displayed in that dialog, clear this check box. If you do not want to include the current setup prerequisite in the list, select this check box. The setup prerequisite is hidden from the list of setup prerequisites, but it is still installed on the target system if needed.  ***Note*** • *Feature prerequisites are never listed in the setup prerequisite dialog at run time, regardless of whether the* ***The prerequisite should be hidden from the installation list*** *check box is selected or cleared. That is, if you add an InstallShield prerequisite to your project and associate it with a feature, that feature prerequisite is not listed in the setup prerequisite dialog that is displayed at run time before the main installation runs.* *Note that if a setup prerequisite or a feature prerequisite is marked as hidden, it is launched with its silent command-line parameters—not its standard command-line parameters. To learn more, see Specifying Command-Line Parameters for an InstallShield Prerequisite.*
Progress should be shown in the prerequisites window (raw MSI file only)	If you want the progress of the installation for the current InstallShield prerequisite to be displayed at run time, select this check box. The progress can be shown only if the InstallShield prerequisite installation is an .msi file; it cannot be shown if the file that the InstallShield prerequisite installation launches is a **Setup.exe** file. In addition, the installation that contains the prerequisite must be a Basic MSI or InstallScript MSI installation. If your InstallShield prerequisite installation is a **Setup.exe** file, this setting is ignored. In addition if the installation that contains the InstallShield prerequisite is an InstallScript installation, this setting is ignored.  ***Note*** • *If you select this check box so that progress can be shown, only some command-line parameters are supported. Command-line parameters are specified on the Application to Run tab of the InstallShield Prerequisite Editor. For more information, see Specifying Command-Line Parameters for an InstallShield Prerequisite.* *Also note that if you specify that the progress should be shown for a prerequisite that is an .msi package, the user interface of the prerequisite's .msi package is not displayed by default. If you want to override this behavior, you can specify* `/qf` *as a command-line parameter on the Application to Run tab.*

Table 12-3 • Settings on the Behavior Tab (cont.)

Setting	Description
If, after installing the prerequisite, the conditions still indicate it is required	If the conditions still indicate that the InstallShield prerequisite needs to be installed after the InstallShield prerequisite installation has been run, either of the following may be true: • The InstallShield prerequisite was not successfully installed. • The conditions that were specified for the InstallShield prerequisite were not accurate. For example, a condition may indicate that the InstallShield prerequisite needs to be installed if a particular file does not exist on the target machine. If the file is still missing even after the InstallShield prerequisite is installed, it is possible that the condition should not have been created. Specify the behavior that should occur if the conditions still indicate that the InstallShield prerequisite needs to be installed. The available options are: • **Abort the setup**—If your application should not be installed, select this option. • **Ask whether to continue the setup**—If the installation should display a message box that prompts the end user to specify whether the main installation should continue, select this option. • **Continue the setup**—If the installation should essentially ignore the InstallShield prerequisite's unmet condition and proceed to the next InstallShield prerequisite installation (if one is required) or the main installation, select this option.

Table 12-3 • Settings on the Behavior Tab (cont.)

Setting	Description
If the prerequisite requires a reboot	Installing an InstallShield prerequisite may require that the target machine be restarted, as described in Restarting a Target Machine for InstallShield Prerequisite Installations. Specify the behavior that should occur if the InstallShield prerequisite requires that a target machine be restarted. • **Exit and resume on reboot**—To end the InstallShield prerequisite installation, allow the InstallShield prerequisite to restart the target machine, and then continue the installation, select this option. • **Note it, fail to resume if the machine is rebooted, and reboot after the installation**—To note that the target machine should be restarted if it is required, to schedule the restart for the end of the main installation, but to exit if the target machine restarts after the InstallShield prerequisite installation, select this option. Thus, if it appears that a restart is required but you want to postpone it until the end of the main installation (or until a subsequent InstallShield prerequisite triggers a restart), select this option. At run time, the value of the Windows Installer property **ISSCHEDULEREBOOT** is set to 1 in the main installation when a restart is still pending. • **Ignore it, and fail to resume if machine is rebooted**—To continue the installation but end it if the target machine is restarted, select this option. Thus, if it appears that a restart is required but you want to try to skip it, select this option. • **Prompt the user only if no reboot is detected, but always reboot the machine and resume on reboot**—To prompt the end user to restart the target machine only if it appears that the target machine does not need to be restarted, select this option. The target machine is restarted if the end user agrees to allow it, and then the installation continues. • **Prompt the user to reboot the machine even if nothing is detected, and resume on reboot**—To prompt the end user to restart the target machine even if it appears that the target machine does not need to be restarted, select this option. The target machine is restarted if the end user agrees to allow it, and then the installation continues. • **Reboot the machine and resume on reboot**—To restart the target machine and then continue the installation, select this option.

Dependencies Tab

When you are creating a new InstallShield prerequisite or modifying an existing one, you can specify other InstallShield prerequisites (.prq files) on which this InstallShield prerequisite depends. The Dependencies tab of the InstallShield Prerequisite Editor is where you specify the dependencies.

When you click the Add button on this tab to add a dependency, the New Dependency dialog box opens.

Errors and Warnings

The following topics in the InstallShield Help Library provide information about errors and warnings that might occur when you are working with your installation.

Table 12-1 • List of Error and Warning Topics

Topic	Description
Build Errors and Warnings	Lists errors and warnings that might occur during the build process.
Media Build Errors and Warnings	Lists InstallScript compiler errors and warnings.
MSI/MSM Conversion Errors	Lists errors that might occur when you use the Open MSI/MSM Wizard to convert an installation package (.msi file) or merge module (.msm file) to an InstallShield project (.ism file).
Windows Installer Run-time Errors	Lists Windows Installer errors that might occur during run time. The errors are related to the built-in InstallShield custom actions that are added automatically to InstallShield projects to support some types of functionality.
Setup.exe Return Values and Run-Time Errors (InstallScript Projects)	Lists the errors that might occur when **Setup.exe** runs for InstallScript installations.
Setup.exe Return Values and Run-Time Errors (Basic MSI and InstallScript MSI Projects)	Lists the errors that might occur when **Setup.exe** runs for Basic MSI and InstallScript MSI installations.
Setup.exe Return Values and Run-Time Errors (Advanced UI and Suite/Advanced UI Projects)	Lists return codes, errors, and logged status codes that might occur when an Advanced UI or Suite/Advanced UI **Setup.exe** installation is run.
Visual Studio Project Import Errors and Warnings	Lists the errors and warnings that might occur when you do any of the following: • Convert a Visual Studio setup project (.vdproj) to an InstallShield Basic MSI project (.ism). • Convert a Visual Studio merge module project (.vdproj) to an InstallShield Merge Module project (.ism). • Import a Visual Studio setup or merge module project (.vdproj) into an InstallShield Basic MSI or Merge Module project (.ism).
Upgrade Errors (Upgrading from InstallShield Professional)	Lists errors that might occur when you upgrade a project created with InstallShield Professional.
Upgrade Warnings (Upgrading from InstallShield Professional)	Lists warnings that might occur when you upgrade a project created with InstallShield Professional.

Table 12-1 • List of Error and Warning Topics (cont.)

Topic	Description
Upgrade Errors and Warnings (Upgrading from InstallShield—Windows Installer Edition)	Lists errors and warnings that might occur when you upgrade a project (.ism file) created with a version of InstallShield—Windows Installer Edition.
HRESULT Values for Windows Installer Run-time Errors	Provides descriptions for InstallShield custom HRESULT codes provided with Windows Installer errors 1904 and 1905.
DIFxAPI Errors (InstallScript Projects)	Describes each of the DIFxAPI errors that can be returned when DIFx driver functions are called.
"String PRODUCT_NAME was not found in string table" Error	Provides description of when this error can occur and what steps you should take to correct it.
InstallScript Error Information	Provides descriptions for InstallScript errors and warnings.
Virtualization Conversion Errors and Warnings	Lists errors and warnings that might occur during the build process when you are creating a virtual application.

In-depth articles about errors and warnings are available in the Knowledge Base.

Build Errors and Warnings

The following table contains a listing of errors and warnings that may occur during the build process.

Note • *You can find detailed information—including resolution information—on most InstallShield build errors and warnings in the Knowledge Base.*

For troubleshooting information about errors and warnings that may occur when you are building a virtual application, see Virtualization Conversion Errors and Warnings.

Table 12-2 • Build Errors and Warnings

Error or Warning Number	Message	Troubleshooting Information
-32000	Build canceled by the user.	This error occurs only when the build is terminated during the build process.
-7355	The %4 value for string %2 does not meet validation criteria for table %1 column %3.	This warning occurs if a localized string value does not meet validation criteria for a column in the String Editor table. To resolve this warning, update the flagged value in the String Editor.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7354	The %4 value for string %2 does not contain a legitimate value for table %1 column %3.	This error occurs if a localized string value does not contain a legitimate value in the named column of the String Editor table. To resolve this error, update the flagged value in the String Editor.
-7350	An error occurred building the ISWSEWrap table. The File %1 could not be found.	This error occurs if your project includes a WiseScript custom action but the executable file that is specified for it in the Custom Actions and Sequences view could not be found. To resolve this error, modify the path in the WiseScript Executable setting for this custom action.
-7348	Error 0x%08x saving file %s	In this error message, %s refers to the path and file name of a file that needs to be saved at build time for an Advanced UI or Suite/Advanced UI project. To resolve this error, ensure that the file is available and is not read-only or locked.
-7347	InstallShield does not support signing with .spc and .pvk files. Replace the digital certificate information in your project with a .pfx certificate, or with a reference to a certificate in a certificate store.	This error occurs if you added .spc and .pvk files to your project in an earlier version of InstallShield that included support for digitally signing with those files, and then you open your project in InstallShield 2016 and attempt to build a release or a patch. To resolve this error, remove the .spc reference from the release or patch configuration. You can replace it with a .pfx certificate, or with a reference to a certificate in a certificate store. To learn more, see: • Digital Signing and Security • Digitally Signing a Release and Its Files at Build Time • Signing a Patch Package • Signing a QuickPatch Package
-7346	The files in this release are being signed with a SHA-1 certificate. Windows will not trust files that were signed with SHA-1 certificates if they were signed after January 1, 2016.	This warning occurs if your project is configured to use a SHA-1 certificate to digitally sign your release at build time. SHA-256 is favored over SHA-1, which is being deprecated because of the potential for security vulnerabilities. Thus, it is recommended that you replace any SHA-1 certificates in your InstallShield projects with SHA-256 certificates. In InstallShield, to replace a SHA-1 certificate with a SHA-256 certificate for signing your releases, use the Signing tab in the Releases view to replace the reference to the current certificate with one for a SHA-256 certificate. To learn more, see Digital Signing and Security.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7337	Feature %1 has no associated packages.	If you build a release for a Suite/Advanced UI project or an Advanced UI project, and the project contains a feature that does not contain any packages, this build warning is displayed. To resolve this build warning, ensure that at least one package is associated with the feature. To learn more, see Associating a Package in an Advanced UI or Suite/Advanced UI Project with a Feature. If you intentionally are not including a package with the feature that is identified in the build warning, you can ignore this build warning.
-7336	Advanced UI projects do not have support for Events.	If you use the Professional edition of InstallShield to open a Suite/Advanced UI project that includes one or more actions in the Events view, InstallShield opens the project as an Advanced UI project, and it displays this build error at build time. Advanced UI projects do not have support for events. To resolve this build error, remove the actions from your project, or use the Premier edition of InstallShield to edit the project and build releases. To learn more, see Using Actions to Extend the Behavior of a Suite/Advanced UI Installation.
-7335	The project contains a reference to undefined Action '%1'.	If you add an action to your Suite/Advanced UI project, associate the action with a package in your project, and then delete part of the action from your project by editing the .issuite project file in a text editor, this error occurs at build time. To resolve this error, remove the action that is referenced in the build error from your .issuite file.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7334	The package '%1' includes references to Windows features, which are not supported in this edition of InstallShield. These references will not be included in the build.	If you use the Professional edition of InstallShield to open a Suite/Advanced UI project that includes support for enabling one or more Windows roles or features, InstallShield opens the project as an Advanced UI project, and it displays this build error at build time. Advanced UI projects do not have support for enabling Windows roles or features. To resolve this build error, remove the support for Windows roles and features from your project, or use the Premier edition of InstallShield to edit the project and build releases. To learn more, see Enabling Windows Roles and Features During a Suite/Advanced UI Installation.
-7329	Including IA64 %1 %2 in an x64 package.	This build warning occurs if the following conditions are true: • x64 is specified for the Template Summary setting. • The project includes an Itanium 64-bit file that the installation may transfer to target systems at run time. An x64 target system cannot run an Itanium 64-bit file. Depending on your requirements, you may want to replace the IA64 file with an x64 file. In other scenarios, you may want to ignore this build warning. For example, if the file's component has a condition that prevents the component from being installed on x64 systems, you may want to ignore this build warning.
-7328	Including x64 %1 %2 in an IA64 package.	This build warning occurs if the following conditions are true: • IA64 is specified for the Template Summary setting. • The project includes an x64 file that the installation may transfer to target systems at run time. An Itanium 64-bit target system cannot run an x64 file. Depending on your requirements, you may want to replace the x64 file with an IA64 file. In other scenarios, you may want to ignore this build warning. For example, if the file's component has a condition that prevents the component from being installed on IA64 systems, you may want to ignore this build warning.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7327	Including 64-bit %1 %2 in a strict 32-bit package.	This build warning occurs if the following conditions are true: • Intel is specified for the Template Summary setting. • Strict is selected for the Platform Validation setting. • The project includes an x64 file that the installation may transfer to target systems at run time. An x86 target system cannot run an x64 file. Depending on your requirements, you may want to replace the x64 file with an x86 file. In other scenarios, you may want to ignore this build warning. For example, if the file's component has a condition that prevents the component from being installed on x86 systems, you may want to ignore this build warning. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.
-7326	Including 32-bit %1 %2 in a strict 64-bit package.	This build warning occurs if the following conditions are true: • An x64 architecture is specified for the Template Summary setting. • Strict is selected for the Platform Validation setting. • The project includes an x86 file that the installation may transfer to target systems at run time. In many cases, an x64 target system can run an x86 file. However, if the x64 system does not have 32-bit Windows-on-Windows (WOW64) support, it may not be able to run an x86 file. Depending on your requirements, you may want to replace the x86 file with an x64 file. In other scenarios, you may want to ignore this build warning. For example, if the file's component has a condition that prevents the component from being installed on x64 systems, you may want to ignore this build warning. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7325	An unexpected error occurred validating the architecture for file %1.	This build error occurs if InstallShield cannot determine whether a file in the project is 32 bit or 64 bit when performing architecture validation. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.
-7324	%1 Custom Action %2 cannot call a function in native code %3.	This build error occurs if your project contains a managed-code custom action for a standard DLL file. To resolve this error, replace the custom action with a standard DLL custom action that calls a function in the DLL file. Otherwise, replace the DLL file with a .NET assembly that was written in managed code such as Visual Basic .NET or C#. For more information, see Calling a Public Method in a Managed Assembly.
-7323	%1 Custom Action %2 cannot call a function in managed code %3.	This build error occurs if your project contains a standard DLL custom action for a .NET assembly file. To resolve this error, replace the custom action with a managed-code custom action that calls a public method in the .NET assembly. Otherwise, replace the .NET assembly with a standard DLL that was written in native code such as C++. For more information, see Calling Functions in Standard DLL Files.
-7322	IA64 %1 Custom Action %2 cannot execute in an x64 package.	This build warning occurs if the following conditions are true: • x64 is specified for the Template Summary setting. • The project contains a custom action file that targets Itanium 64-bit systems. An x64 target system cannot run an IA64 custom action. To learn more, see Using the Template Summary Property.
-7321	x64 %1 Custom Action %2 cannot execute in an IA64 package.	This build warning occurs if the following conditions are true: • IA64 is specified for the Template Summary setting. • The project contains a custom action file that targets x64 systems. An IA64 target system cannot run an x64 custom action. To learn more, see Using the Template Summary Property.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7320	64-bit %1 Custom Action %2 must not be included in a strict 32-bit package.	This build error occurs if the following conditions are true: • Intel is specified for the Template Summary setting. • Strict is selected for the Platform Validation setting. • The project includes a 64-bit custom action file. Note that 64-bit DLL files cannot be run on 32-bit target systems. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.
-7319	32-bit %1 Custom Action %2 must not be included in a strict 64-bit package.	This build error occurs if the following conditions are true: • x64 is specified for the Template Summary setting. • Strict is selected for the Platform Validation setting. • The project includes a 32-bit DLL custom action file. Note that 32-bit DLL files cannot be run on 64-bit target systems that do not have 32-bit Windows-on-Windows (WOW64) support. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.
-7318	InstallScript Custom Action %1 must not be included in a strict 64-bit package.	This build error occurs if the following conditions are true: • x64 is specified for the Template Summary setting. • Strict is selected for the Platform Validation setting. • The project includes an InstallScript custom action. Note that InstallScript custom actions cannot be run on 64-bit target systems that do not have 32-bit Windows-on-Windows (WOW64) support. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7317	64-bit VBScript/JScript Custom Action %1 must not be included in a strict 32-bit package.	This build error occurs if the following conditions are true: • Intel is specified for the Template Summary setting. • Strict is selected for the Platform Validation setting. • The project includes a 64-bit VBScript or JScript custom action file. Note that 64-bit script files cannot be run on 32-bit target systems. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.
-7316	32-bit VBScript/JScript Custom Action %1 must not be included in a strict 64-bit package.	This build error occurs if the following conditions are true: • x64 is specified for the Template Summary setting. • Strict is selected for the Platform Validation setting. • The project includes a 32-bit VBScript or JScript custom action file. Note that 32-bit script files cannot be run on 64-bit target systems that do not have 32-bit Windows-on-Windows (WOW64) support. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.
-7315	An unexpected error occurred validating the architecture for custom action %1.	This build error occurs if InstallShield cannot determine whether a custom action file in the project is 32 bit or 64 bit when performing architecture validation. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.
-7314	An unexpected error occurred validating the architecture for custom actions.	This build error occurs if InstallShield cannot determine whether a file in the project is 32 bit or 64 bit when performing architecture validation. For more information, see Selecting the Appropriate Type of Architecture Validation for Builds.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7313	Advanced UI projects do not have support for UWP app packages.	If you open a Suite/Advanced UI project that includes a UWP app package in the Professional edition of InstallShield, it is opened as an Advanced UI project, and it displays this build error at build time. Advanced UI projects do not have support for UWP app packages. To resolve this build error, remove the UWP app package from your project, or use the Premier edition of InstallShield to edit the project and build releases.
-7312	The certificate file '%1' is selected for package '%2', but it is not included with this package's files. Either delete the file from the Certificate File setting, or add the file to the UWP app package.	The versions of Windows that support UWP app packages do not install a sideloading package unless they trust the source of the package. If you are using a custom certificate, such as the one that is generated in Visual Studio, the Suite/Advanced UI installation can make Windows trust the source of your package by adding the custom certificate to the certificate store of target systems. The Suite/Advanced UI installation does this if you include the .cer file in your sideloading package, and specify it in the Certificate File setting for the UWP app package in the Packages view. If you specify a .cer file but do not include the file in your project, you may encounter this build warning at build time. To add the .cer file, find the UWP app package in the Packages view, and expand its node in the view. Select the Packages Files node, and right-click the right-pane, and then click Add. InstallShield lets you browse to select the file that you want to add.
-7311	Unable to load manifest file for sideloading app package '%1'.	This build error occurs if you are trying to build a Suite/Advanced UI installation that includes a sideloading UWP app package but InstallShield cannot read the app package's manifest file. This may occur if you are trying to build on a Windows XP or Windows Server 2003 system. The ability to create and build a Suite/Advanced UI installation that includes a sideloading UWP app package requires Windows Vista or later or Windows Server 2008 or later on the machine that has InstallShield or the Standalone Build. To resolve this error, remove the UWP app package from the Packages view, or build your release on Windows Vista or later or on Windows Server 2008 or later.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7308	Suite/Advanced UI installations do not support sideloading UWP app packages that are built for only ARM processors.	This error occurs if you include a UWP app package that targets the ARM processor architecture in a Suite/Advanced project. InstallShield does not have support for including that type of package in an installation. To resolve this error, use the Packages view remove the package that is listed in the error message from your project, and consider replacing it with a package that targets x86 or x64 systems, or that is neutral.
-7295	Advanced UI projects do not have support for .exe primary packages.	InstallShield lets you add .exe packages as a primary package to a Suite/Advanced UI project, which is available in the Premier edition of InstallShield; however, InstallShield does not let you add .exe packages as primary packages to an Advanced UI project, which is available in the Professional edition of InstallShield. If you use the Professional edition of InstallShield to open a project that was created in the Premier edition, and if the project contains an .exe package as a primary package, this build error occurs. To resolve this build error, either use the Premier edition of InstallShield to build the release, or use the Packages view to remove the .exe package from the project. For more information about the differences between Advanced UI and Suite/Advanced UI projects, see Advanced UI Projects vs. Suite/Advanced UI Projects.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7294	An Advanced UI project can include only one language.	A Suite/Advanced UI project, which is available in the Premier edition of InstallShield, includes support for multiple languages; however, an Advanced UI project, which is available in the Professional edition of InstallShield, includes support for only one. If you use the Professional edition of InstallShield to open a project that was created in the Premier edition, and if the project contains more than one language, this build error occurs. To resolve this build error, either use the Premier edition of InstallShield to build the release, or remove all but one language from the Setup Languages setting in the General Information view of the project. You may also need to remove all but one language from the UI Languages setting on the Build tab in the Releases view. For more information about the differences between Advanced UI and Suite/Advanced UI projects, see Advanced UI Projects vs. Suite/Advanced UI Projects.
-7293	An Advanced UI project can include only one primary package.	A Suite/Advanced UI project, which is available in the Premier edition of InstallShield, includes support for multiple primary packages; however, an Advanced UI project, which is available in the Professional edition of InstallShield, includes support for only one. If you use the Professional edition of InstallShield to open a project that was created in the Premier edition, and if the project contains more than one primary package, this build error occurs. To resolve this build error, either use the Premier edition of InstallShield to build the release, or remove all but one primary package from the Packages view of the project. For more information about the differences between Advanced UI and Suite/Advanced UI projects, see Advanced UI Projects vs. Suite/Advanced UI Projects.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7292	An update URL is set but this release is not configured to be digitally signed.	This build warning occurs if a URL is specified in the Update URL setting on the Setup.exe tab in the Releases view of the Advanced UI or Suite/Advanced UI project, but the release is not configured to be digitally signed. To resolve this warning, specify digital signature information on the Signing tab for the release in the Releases view. To learn more about updates, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.
-7291	An update URL is set but the project contains packages with files located on the source media.	This warning occurs if a URL is specified in the Update URL setting on the Setup.exe tab in the Releases view of the Advanced UI or Suite/Advanced UI project, but one or more packages in the Advanced UI or Suite/Advanced UI project is configured to be stored on the source media. If you are building a base Advanced UI or Suite/Advanced UI setup launcher, you can ignore this warning. If you are building the update Advanced UI or Suite/Advanced UI setup launcher that will be downloaded and launched when an earlier base setup launcher is run on a target system, change the run-time location of the packages in your project to either extracted from the setup launcher or downloaded from the Web. The update setup launcher cannot rely on packages that are stored on the source media. For more information, see Specifying a Run-Time Location for a Specific Package in an Advanced UI or Suite/Advanced UI Project. To learn more, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.
-7256	To digitally sign a software identification tag file, .NET Framework 3.5 must be installed.	If your project is configured to include a software identification tag and if the release that you are building is configured to use a .pfx file to digitally sign your release, InstallShield attempts to digitally sign the tag that it creates at build time. If your build machine does not have .NET Framework 3.5, this build warning is displayed. To resolve this build warning, install .NET Framework 3.5 on your build machine.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7255	Only .pfx files can be used to digitally sign a software identification tag file.	If your project is configured to include a software identification tag and if the release that you are building is configured to use a certificate in a certificate store to digitally sign your release, InstallShield generates this build warning. To resolve this build warning, switch from a certificate in a certificate store to a .pfx file. If you do not need to have a digitally signed tag, you can ignore this warning.
-7246	Package "%1" has an MSI Package condition in which an asterisk (*) is used as a placeholder for one of the condition's settings. InstallShield cannot substitute this placeholder because the appropriate .msi package could not be found.	In order for InstallShield to replace an asterisk in a condition with the appropriate information (such as the product code or product version) from a package in an Advanced UI or Suite/Advanced UI project, the package must be available at build time. To resolve this error, ensure that the package file is in the source location. If dynamic file linking is used for the package, ensure that the dynamic link filters are not excluding the package file from the build.
-7244	The project does not contain any primary packages. You may need to add a package, or select Primary in the Package Type setting for a package in the Packages view.	This error occurs if your Advanced UI or Suite/Advanced UI project does not contain any packages that are configured to be the primary package. To resolve this error if your project does not contain any packages, use the Packages view to add a package to your project. For more information, see Guidelines for Adding Packages to an Advanced UI or Suite/Advanced UI Project. If you already have a dependency package in your project and you want to make it a primary package, use the Package Type setting in the Packages view. To learn more, see Primary Packages vs. Dependency Packages in Advanced UI and Suite/Advanced UI Projects.
-7243	The Detection Condition setting for package '%1' in the Packages view is empty. This type of condition evaluates whether the package is already installed on target systems.	The Detection Condition setting cannot be empty for an .exe type of package in an Advanced UI or Suite/Advanced UI project. For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7240	Package '%1' is not associated with any features.	Each package that is included in the Packages view of an Advanced UI or Suite/Advanced UI project should be associated with one or more features. Otherwise, the package are not included in any releases that you build. For more information, see Associating a Package in an Advanced UI or Suite/Advanced UI Project with a Feature.
-7239	The target file '%1' that was selected for an operation's Target setting in the Packages view for package '%2' was not found in the project.	This error occurs if the target file for one of the packages in the Advanced UI or Suite/Advanced UI installation is missing at build time. To resolve this error, ensure that the target file is in the source location. If dynamic file linking is used for the package, ensure that the dynamic link filters are not excluding the target file from the build.
-7236	InstallShield could not create the software identification tag because the value for the %1 setting in the General Information view is invalid. Ensure that it does not contain any of the following characters: \ / : * ? " < > |	InstallShield uses the values that you enter for the Unique ID and Tag Creator ID settings as part of the name of the tag file (*TagCreatorID_UniqueID*.swidtag). Therefore, the ID that you enter must not contain any characters that are invalid for file names. To resolve this issue, enter a valid value in the setting that is mentioned in the warning message. For more information, see Including a Software Identification Tag for Your Product.
-7235	InstallShield could not create the software identification tag because the %1 setting in the General Information view is empty.	Creation of a software identification tag requires that several settings in the General Information view have values. To resolve this warning, enter the appropriate value in the setting that is mentioned in the warning message, or select No for the Use Software Identification Tag setting. For more information, see Including a Software Identification Tag for Your Product.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7234	InstallShield could not create the software identification tag because the template file **Swidtag.xml** could not be opened.	InstallShield uses the **Swidtag.xml** file, which is installed to one of the following locations, to build tag files: *InstallShield Program Files folder***Support\0409** *InstallShield Program Files folder***Support\0411** To resolve this issue, ensure that the **Swidtag.xml** file is present and not locked. For more information, see Including a Software Identification Tag for Your Product.
-7233	An invalid URL (%2) is specified in the Packages view for package %1.	This build error occurs if you selected Download From The Web for a package in the Packages view of an Advanced UI or Suite/Advanced UI project, but the URL setting under the Location setting is blank, or it contains an invalid URL. To resolve this error, enter a URL that starts with one of the following strings: • http:// • https:// • ftp://
-7225	The project does not contain any packages.	This build error occurs if try to build an Advanced UI or Suite/Advanced UI project that does not contain any packages. To resolve this error, use the Packages view to add a package to your project. For more information, see Guidelines for Adding Packages to an Advanced UI or Suite/Advanced UI Project.
-7223	A Mode condition or a Detection condition is required for the Setup.exe to run correctly.	This build error occurs if your Advanced UI or Suite/Advanced UI project includes an .exe package but you have not configured the appropriate conditions for it yet. For information on adding detection conditions to a package in a project, see the following: • Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects • Guidelines for Defining Conditions in an Advanced UI or Suite/Advanced UI Project For information on how mode conditions are created, see Triggering Install Mode or Maintenance Mode of an Advanced UI or Suite/Advanced UI Installation.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7222	No supported languages included in the media. You must select at least one supported language for the media.	This error occurs only when no supported language is selected in the media. To resolve this issue, ensure that at least one language is selected for the Language(s) setting in the Releases view.
-7220	InstallShield encountered an error while including a DIM reference.	This error occurs if your project includes a DIM reference, as shown in the DIM References view, but InstallShield cannot find the DIM project file (.dim). This might occur if the .dim file was moved to a different location or it was renamed, but the path or file name in the installation project that contains the DIM references has not been updated.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7218	Conflicts were detected building DIMs. In Table %1 Row %2, data files differ in the following columns: %3.	This warning occurs if InstallShield encounters a data conflict between the installation project that is open and one of its DIM references. In this warning message, %1 indicates the name of the table that contains the conflict, and %2 indicates the applicable row; %3 is in the following format: **ColumnName ("InstallationUnitValue" <> "InstallationProjectValue")** The table data for the DIM reference is available in the Direct Editor view of the DIM project. The table data for the installation is available in the Direct Editor view of the installation project. If a data conflict occurs, InstallShield uses either the installation project data or the DIM data; it depends on which value is selected in the Conflict Resolution setting on the Build Options tab in the DIM References view: ● **Use Base Project Value**— If this value is selected, InstallShield uses the value that is in the installation project instead of the value that is in the DIM project. ● **Use DIM Project Value**—If this value is selected, InstallShield uses the value that is in the DIM project instead of the value that is in the installation project. If the appropriate value is being used, you can ignore this warning message. If you want to use the value that is being overridden, you can either revise the value in that project, or change the value of the Conflict Resolution setting. Note, however, that if other conflicts occur for this DIM reference, changing the value of the Conflict Resolution setting may also change the resulting built values for those other conflicts. For more information, see Resolving Build-Time Conflicts Between a Basic MSI Project and a DIM Reference.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7216	This project includes InstallScript objects that are deprecated. These objects should be removed from the project. A prerequisite should be used instead where possible.	InstallScript objects have been deprecated in favor of InstallShield prerequisites. The recommended alternative for InstallScript objects is InstallShield prerequisites. You can use the InstallShield Prerequisite Editor to create your own InstallShield prerequisites. You can share these InstallShield prerequisites among InstallScript, InstallScript MSI, and Basic MSI projects. To learn more, see: • Defining InstallShield Prerequisites • Adding InstallShield Prerequisites, Merge Modules, and Objects to InstallScript Projects
-7215	Skipping build events. Build events are supported only in the Premier edition of InstallShield.	InstallShield lets you specify commands that you want to be run at various stages of the build process. This build event functionality is available in the Premier edition of InstallShield. For more information, see Specifying Commands that Run Before, During, and After Builds. If you try to use the Professional edition of InstallShield to build a release that includes build events, InstallShield ignores the build event settings and generates this warning. To resolve this warning, use the Premier edition of InstallShield to build the release.
-7214	Error loading table %1.	This error occurs if the table that is mentioned in the error message is missing from your project, or it is missing columns. To resolve this error, open the Direct Editor view, and look for the table that is mentioned in the error message. If it exists, compare the columns in the table with those in a new project, and add any missing tables. If the table does not exist, you can export it from a new project as an .idt file, and then import the .idt file into your existing project. For more information, see: • Exporting Tables from the Direct Editor • Importing Tables with the Direct Editor

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7211	Building a compressed Network Image Setup.exe. All other build types are not allowed in the evaluation mode of InstallShield.	If you are using InstallShield in evaluation mode (that is, you have not activated it), you can build compressed **Setup.exe** files, but no other release types. If you are creating a Windows Installer–based release, the .msi database is always embedded in the **Setup.exe** file. If you try to build an uncompressed release in evaluation mode, the uncompressed option is ignored, and InstallShield displays this build warning. Once you have activated InstallShield, the evaluation-mode limitations are removed.
-7209	The current Windows Installer package name includes Unicode characters that may cause a run-time error if a compressed Network Image Setup.exe is built.	This build warning occurs if the following conditions are true: • For the name of the .msi package, you include Unicode characters that are not supported by your build machine's ANSI code page. • You include a setup launcher (**Setup.exe** file) with your release. This warning occurs to inform you that when the Windows Installer package is extracted from **Setup.exe** to a temporary location on the target system, run-time error 1152 may occur. To resolve this warning, change the name of the .msi package so that it does not contain Unicode characters that are not supported by your machine's ANSI code page. To do so, you can use the MSI Package File Name setting on the General tab for a product configuration in the Releases view.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7208	The current location for this package includes Unicode characters that may cause a run-time error when Windows Installer tries to extract its .cab files.	This warning occurs if the following conditions are true: • The path for the release that InstallShield is building contains Unicode characters that are not supported by your build machine's ANSI code page. • You configure the release so that your product's program files are compressed into .cab files. You can ignore this warning if end users will launch your release from a location whose path does not contain Unicode characters that are not supported by your build machine's ANSI code page. If end users try to launch your installation from a path that contains Unicode characters that are not supported by the target system's ANSI code page, run-time error 1311 occurs when Windows Installer tries to extract the .cab files.
-7207	A package cannot contain both the MsiLockPermissionsEx and LockPermissions tables.	An installation should not contain both the **MsiLockPermissionsEx** table and the **LockPermissions** table. If you use the Permissions setting for a service, and the settings under it, you are configuring the **MsiLockPermissionsEx** table of the package. If Traditional Windows Installer handling is selected for the Locked-Down Permissions setting in the General Information view and you have set permissions for one or more files, folders, or registry keys in your project, you are configuring the **LockPermissions** table of the package. For more information, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.
-7201	The language '%1' is missing from this project but included in this release.	This build warning occurs if a language is selected in the UI Languages setting on the Build tab in the Releases view, but it is not also selected in the Setup Languages setting in the General Information view. This build warning is alerting you that the release will not include the specified run-time language. If you do not want to include the specified run-time language in your release, you can ignore this warning. If you do want to include the specified run-time language in your release, add the language to the Setup Languages setting.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7187	The Find What setting for the %1 text change is not configured. This text change will not be performed.	You have added a change item to a change set in the Text File Changes view; however, the value for the Find What setting is blank. This setting must have a value; otherwise, the text file changes are not made at run time. For more information, see Specifying a Change for a Text File.
-7186	The Include Files setting for the %1 text change set is not configured. None of the text changes associated with this text change set will be performed.	You have added a change set in the Text File Changes view; however, the value for the Include Files setting is blank. This setting must have a value; otherwise, the text file changes are not made at run time. For more information, see Creating a Text Change Set.
-7185	The %1 translation for string identifier %2 includes characters that are not available on code page %3.	This error occurs if both of the following conditions are true: • The string identifier value that you entered for the specified language contains characters that are not available in that language's code page. • You are building a language-specific (ANSI) database. To resolve this error, consider using the String Editor view to edit the value of the string identifier so that it uses characters from the appropriate code page. If you must use characters that are not available in the target language's code page, consider selecting Yes for the Build UTF-8 Database setting. Note, however, that this may result in user interface issues in some scenarios because the Windows Installer does not fully support UTF-8 databases. For more information, see the description of the Build UTF-8 Database setting.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7184	The %1 column of the %2 table includes characters that are not available on code page %3: "%4"	This error occurs if both of the following conditions are true: • A table in the source project database contains characters that are not available in the code page for one or more target languages. • You are building a language-specific (ANSI) database. To resolve this error, consider changing the data that is described in the error message so that it uses characters from the appropriate code page. For example, if the error message mentions the **Shortcut** table, consider changing the string in the Shortcuts view. If you must use characters that are not available in the target language's code page, consider selecting Yes for the Build UTF-8 Database setting. Note, however, that this may result in user interface issues in some scenarios because the Windows Installer does not fully support UTF-8 databases. For more information, see the description of the Build UTF-8 Database setting.
-7174	"%1" contains a reference to a string table entry '%2' that is not included in the project's string table.	In this warning message, %1 indicates the fully qualified path to an InstallScript file (.rul) in the project, and %2 indicates the string ID that is used in that .rul file but is not listed as one of the project's string entries. InstallShield displays this warning at build time for each string identifier that is found in the project's InstallScript file but not in the String Editor view. To see the line in the InstallScript code that has the string identifier, double-click the warning message that is displayed on the Tasks tab of the Output window at build time. The string identifier is preceded by the string constant operator (@). To resolve this build issue, ensure that the string identifier is defined in the String Editor view. In addition, ensure that the spelling of the string identifier in the InstallScript code matches that in the String Editor view.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7143	Component %1 installs to a 64-bit folder but is not marked as a 64-bit component. This may result in an incorrect installation path for this component's files.	If a component is not configured to be 64 bit, Windows Installer may not install the component's files to the appropriate 64-bit location. To specify that a component is 64 bit, select Yes for the component's 64-Bit Component setting. For more information about the 64-Bit Component setting, as well as additional component settings, see Component Settings.
-7138	The Msi Command Line of the release currently being built contains REINSTALLMODE. The parameter appears to have the 'a' option which will force all files to be reinstalled, regardless of checksum or version.	Check the Knowledge Base for information about this error, or request technical support.
-7137	The %1 file that you selected for the Help File Path setting in the Custom Actions view for the %2 custom action does not contain any text.	Create a file that describes the intended behavior of the custom action that is mentioned in the warning. The file should be a text-based file such as a .txt, .htm, or .rtf file. Then in the Custom Actions and Sequences view (or the Custom Actions view), select the custom action and select the file that you created for the Help File Path setting. To learn more, see Documenting the Behavior of Custom Actions.
-7135	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-7128	Error embedding setup.exe.manifest.	This error occurs if there is a problem embedding the application manifest in the **Setup.exe** launcher. The application manifest specifies the minimum privilege level required by your installation's **Setup.exe** file for running the installation (the setup launcher, any InstallShield prerequisites, and the .msi file) on Windows Vista and later platforms. This error may occur if the template manifest file is missing from the InstallShield **Support** folder. It also may occur if the **Setup.exe** template is missing from the InstallShield **Redist\Language Independent\i386** folder. To resolve this error, try running a repair of InstallShield.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7125	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-7124	Could not invoke MSBuild. Make sure that the .NET Framework %1 is installed at '%2' on the system. If you are running InstallShield from within Visual Studio, you must use Visual Studio 2005 or later.	If you use MSBuild to build Visual Studio solutions with InstallShield projects, you must have .NET Framework 3.5 or later on your machine. In addition, you must be using Visual Studio 2005 or later.
-7123	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-7122	Error writing information to the source file for %1. Verify that the file is writeable, and\or that the file ISTemplate.manifest is not read only.	Check the Knowledge Base for information about this error, or request technical support.
-7121	The manifest file %1 does not contain COM information in at least one file element. Please ensure that all COM modules associated with the manifest are self-registering and verify that the self-registration process does not fail for each COM module.	Check the Knowledge Base for information about this error, or request technical support.
-7120	Error building table %1	Check the Knowledge Base for information about this error, or request technical support.
-7119	Could not locate a key file for component %1. Advertised shortcut %2 points to the key file of this component.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7118	The InstallScript Custom Action %1 cannot be sequenced in InstallUISequence for an InstallScript MSI project.	The User Interface sequence does not run in InstallScript MSI installations; the event-driven InstallScript code handles the user interface. To resolve this issue, do one of the following: • Schedule the InstallScript custom action for the Execute sequence. • Use InstallScript event–driven code instead of an InstallScript custom action.
-7117	Custom Action %1 called from Standard dll and stored in Binary table cannot have deferred/ rollback/commit execution.	This error occurs if you try to call a deferred-, rollback-, or commit-execution custom action using a standard .dll. When you call a non-Windows Installer .dll (that is, one that does not have the MyFunc(MSIHANDLE) entry point), InstallShield does the following: • InstallShield creates a wrapper .dll with the standard Windows Installer entry point. • InstallShield stores your .dll plus the InstallShield wrapper .dll in the **Binary** table. When the custom action is called, Windows Installer calls the InstallShield wrapper .dll. This, in turn, extracts your .dll from the **Binary** table and calls the function that you specified. However, the InstallShield wrapper .dll does not have access to the **Binary** table during deferred, rollback, or commit custom actions, and thus it cannot extract and call into your .dll. To resolve this error, try one of the following options: • Create a new .dll that has a standard Windows Installer entry point and call it directly. • Use InstallScript code to do perform the same functionality if you do not want to write your own .dll. • Schedule your custom action to run in immediate mode instead of deferred, rollback, or commit mode.
-7116	Custom Action DLL %1 not found in release.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7113	Language support for %1 is not included in this edition.	This error appears if you try to enable too many languages in a project, building with more than two in a non-Premier edition. The build is successful (unless stop at first error is enabled), but the extra languages are not built. The first two qualified languages are built. %1 is replaced with the name of the language that is not included in this release. To resolve this error, use the Premier edition of InstallShield to build your release.
-7108	%s is too large to store in a CAB (2GB maximum).	Check the Knowledge Base for information about this error, or request technical support.
-7107	The merge module %1 silently fails to install from a compressed build. Applying compression post-build may work properly.	Check the Knowledge Base for information about this error, or request technical support.
-7104	The destination of component %1 is the GlobalAssemblyCache. The component must contain a key file.	Check the Knowledge Base for information about this error, or request technical support.
-7101	This component %1 is shared between multiple features, which can create problems for features that are going to use MSI's Install On Demand technology.	Check the Knowledge Base for information about this error, or request technical support.
-7098	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-7097	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-7096	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-7095	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7092	Missing binary entry ISSetup.dll	This error occurs if you build a project that includes an InstallScript custom action, but the InstallScript engine (**ISSetup.dll**) has been removed from the **Binary** table. To resolve this error, manually add a new entry called **ISSetup.dll** to the **Binary** table and browse to the following DLL: *InstallShield Program Files Folder***Redist\Language Independent\i386\ISSetup.dll**
-7088	Failed to load %1. This file needs to be loaded in order to build the associated custom object.	Verify that the file, %1, is in the System subfolder within the InstallShield Program Files folder.
-7087	Custom objects cannot be included with your installation media due to an error creating the Windows Installer Object. In order to include custom objects, Windows Installer needs to be installed on the build system.	Verify that Windows Installer is installed properly on the build system.
-7086	The device driver installation framework is not currently available. The device driver feature will not be fully functional until an update has been provided. Please contact technical support for more details.	This error occurs when you build a release for a project in which you used the Device Driver Wizard to include a device driver package with your installation. The redistributable needed to install the device driver will be available in an update to InstallShield. Instructions for obtaining the update will be posted in an article for this build error in the Knowledge Base when the redistributable is available. To resolve the build error before you receive the redistributable, you can remove the device driver component in the Components view and then rebuild the release.
-7084	The VBScript Custom Action %1 does not point to a valid VBS file.	This warning message is displayed if you add a VBScript custom action to your installation but the file specified for the custom action is not a VBScript file. To resolve this error, select the appropriate type of file for the specified custom action in the Custom Actions and Sequences view (or the Custom Actions view).
-7083	Script '%s' is not associated with any connection.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7082	An error occurred while generating your Sql Script.	Check the Knowledge Base for information about this error, or request technical support.
-7080	An error occurred while testing the component to see if the device driver package scan at build property is set.	Check the Knowledge Base for information about this error, or request technical support.
-7079	An error occurred while getting the path to the device driver package.	Check the Knowledge Base for information about this error, or request technical support.
-7078	Unable to scan device driver package %1. The file is missing.	Check the Knowledge Base for information about this error, or request technical support.
-7077	An error occurred while scanning a device driver package.	Check the Knowledge Base for information about this error, or request technical support.
-7065	The MSI 3.1 Engine could not be found. You can download this using the Redistributable Downloader under the Tools menu, if it has been made available by Microsoft. We have included the 3.0 Engine in your built installation.	This message is displayed as a build error if version 3.1 of the Windows Installer engine was selected to be included but it was not found on the build machine. This message is displayed as a build warning if the version 3.1 or 2.0 best-fit option was selected but version 3.1 of the Windows Installer engine was not found on the build machine.
-7064	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-7063	The File %1 in Component %2 is being installed to the Windows Fonts folder, but there is no corresponding record in the Font table. The Font will not be registered properly on the target system.	To resolve this warning, open the Files and Folders view and remove the font mentioned in the message. Then re-add the font to the Fonts folder; InstallShield adds the corresponding record to the Font table automatically.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7062	Refreshing the COM+ settings from the client machine. An error occurred while refreshing the COM+ settings from the client machine.	This build error occurs when an exception error has occurred with a COM+ application that is selected in the Component Services view. An option is provided to refresh the COM+ settings during the build process, and build error -7062 occurs when this option is selected and the refreshing process has failed. To resolve this error, you can open the Component Services view, select the COM+ application, and clear the Refresh the COM+ settings from the client machine at build check box on the Installation tab.
-7061	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-7060	Due to licensing requirements, InstallShield requires that you provide the MSI 3.0 engine redistributable from the MSI 3.0 Beta. Please copy instmsi.exe from the MSI 3.0 Beta folder to \redist\Language Independent\i386\MSI3.0\instmsiw.exe.	Check the Knowledge Base for information about this error, or request technical support.
-7059	Unable to update the latest patch version property. As a result, future patches built from this project may not be sequenced properly. Make sure the project is not marked as read-only.	Check the Knowledge Base for information about this error, or request technical support.
-7058	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-7036	An error occurred while loading the primary application assembly.	Check the Knowledge Base for information about this error, or request technical support.
-7025	The COM+ component server file '%1' cannot be found on your system. The component '%2' will not be installed on the target system. You will need to install the component manually after your COM+ application is installed.	%1 refers to the COM+ DLL that is missing, and %2 refers to the ProgID. To resolve this warning, add the DLL to your system in the appropriate location, or remove the COM+ component from your project.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7024	To include J# in the build you must also include .NET.	J# requires the .NET Framework Redistributable Package, but it is not included in the installation. If you are sure that your end users already have the .NET Framework on their systems, you can ignore this warning message. However, if there is a chance that they will not have the .NET Framework, add it to your installation. If the .NET Framework is not on their systems, they might have trouble installing J# as part of your installation. For more information, see Adding .NET Framework Redistributables to Projects.
-7023	Internal build error.	This error occurs if a merge module such as MFC 7.1 is not configured correctly or it is corrupted. If you have more than one merge module in your project but you do not know which one is causing the error, you need to first identify the problematic merge module. You can do this by removing a merge module from your project and then building your release. If the error recurs, you have identified the problematic merge module. If not, remove a different merge module and then build. Once you have identified the merge module causing the error, try reinstalling it. If that does not resolve the error, contact the author to obtain the latest version.
-7015	Error copying setup.inx to media folder. Unexpected error.	This error occurs if Setup.inx, the compiled script file, cannot be copied to the Media folder located in the **\Media\\Disk Images\Disk 1** folder. This can occur if you are out of disk space or the file is locked.
-7014	Error copying setup.inx to media folder. Script is read-only.	This error occurs if Setup.inx, the compiled script file, is marked as read-only. To resolve this error, find the Setup.inx file, which is usually in the **\Media\\Disk Images\Disk 1** folder. Right-click this file and select Properties. Then clear the Read-only check box.
-7012	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-7011	%1 must have a strong name to be installed to the GlobalAssemblyCache.	You can install an assembly to the Global Assembly Cache only if it has a strong name. A strong name contains the assembly's simple text name, version number, culture information (if available), a public key, and a digital signature. Using strong-named assemblies in the Global Assembly Cache enables you to have multiple versions of an assembly with the same name but with different versions of .dll files. The assemblies stored in the Global Assembly Cache can be shared by different applications on a computer. To resolve this error message, you have two options. • Change the target destination for the assembly to a location other than the Global Assembly Cache if sharing that assembly with other applications is not explicitly required. Note that it is not necessary to install assemblies into the Global Assembly Cache to make them accessible to COM interop. • Select an assembly that has a strong name for installation to the Global Assembly Cache. Note that once an assembly is created, you cannot sign it with a strong name. You can sign an assembly with a strong name only when you create it.
-7001	During the last build, the .msi file was only partially built. In order to stream the script into the .msi file, you must have a complete .msi file. Build a complete .msi file by selecting "Build <ReleaseName>" from the Build menu.	When you cancel the build, a partially built .msi file exists. If you attempt to compile script into the partial .msi file, this error occurs. Select "Build <ReleaseName>" from the Build menu to build a complete .msi file.
-6654	The file %1 is not the key file of component %2. If you call a function in a standard DLL that is installed with the product and the action is scheduled as deferred, the DLL you are calling must be the component's key file.	Set the called DLL as the component's key file.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6653	The feature %1 in the installation contains more than 1600 components. There is a maximum limit of 1600 components per feature.	For more information, see ICE47 and FeatureComponents Table.
-6652	Could not create _isconfig.xml for use with InstallUtilLib.dll.	Check the Knowledge Base for information about this error, or request technical support.
-6651	The setup you are building contains more than 32,767 files. Automatically switching setup package to appropriate MSI schema.	This warning appears if your installation package contains more than 32,767 files. InstallShield automatically applies the large package schema. ***Caution*** • *There is an issue with patching if your package uses the large package schema. When you build the patch, the following error is triggered: "Can't Generate Primary Transform."*
-6647	Cannot move file from %1 to %2.	Check the Knowledge Base for information about this error, or request technical support.
-6646	The %1 property in the merge module %2 is set to NULL. This property cannot be NULL.	Check the Knowledge Base for information about this error, or request technical support.
-6645	%1 failed to load. Error: %2 Error Description: %3	Check the Knowledge Base for information about this error, or request technical support.
-6641	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6640	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6639	The merge module %1 requires one of the following merge modules also to be included in the setup: %2.	Check the Knowledge Base for information about this error, or request technical support.
-6638	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6637	Invalid registry data for component NewComponent1 while importing *"C:\RegTest\SingleSlashl"=dword: 2* (where the information in italics represents the invalid data).	The invalid data in this example is the use of a single backslash instead of two backslashes. Another example would be an entry with *DWORD* in it instead of *dword*.
-6636	The file key %1 and %2 are found in the File table. Despite having different cases, the identical key names will cause an unexpected result when the files are installed on the target system. This occurs because the compressed files in the cabinet file are named using the file keys. To correct this issue, change one of the file keys to be unique in the cabinet file if you are building a compressed setup or a merge module. You can change the file key name in the Direct Editor view.	Check the Knowledge Base for information about this error, or request technical support.
-6635	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6634	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6633	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6632	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6631	InstallScript MSI projects do not have Multiple Instance support.	Check the Knowledge Base for information about this error, or request technical support.
-6630	The property InstanceId does not appear in the Property table.	Check the Knowledge Base for information about this error, or request technical support.
-6629	The value for the property InstanceId is not numeric.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6628	The value for the property InstanceId is duplicated with a value in the ISProductConfigurationInstance table.	Check the Knowledge Base for information about this error, or request technical support.
-6627	Component %1 has nonfile data but does not have InstanceId in its condition.	Check the Knowledge Base for information about this error, or request technical support.
-6626	Could not add instance transform %1 to substorage.	Check the Knowledge Base for information about this error, or request technical support.
-6625	Could not create instance transform %1.	Check the Knowledge Base for information about this error, or request technical support.
-6624	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6623	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6622	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6620	Could not open the reference package %1 for key synchronization.	Check the Knowledge Base for information about this error, or request technical support.
-6619	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6618	An error occurred while setting the key paths for dynamically created components.	Check the Knowledge Base for information about this error, or request technical support.
-6617	An Error occurred while adding the file '%1' to the synchronization maps.	Check the Knowledge Base for information about this error, or request technical support.
-6616	An error occurred while setting the key path for component %1.	Check the Knowledge Base for information about this error, or request technical support.
-6615	Could not update File.FileName for File %1.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6614	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6613	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6612	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6611	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6610	An error occurred determining if the dynamically linked file %1 is a modified file.	Check the Knowledge Base for information about this error, or request technical support.
-6609	An error occurred determining if the dynamically linked file %1 is a new file.	Check the Knowledge Base for information about this error, or request technical support.
-6608	Your splash screen will not be displayed during the installation because %1 is a compressed bitmap file. To display a splash screen, you must specify a uncompressed bitmap file.	Check the Knowledge Base for information about this error, or request technical support.
-6607	Your splash screen will not be displayed during the installation because %1 is not a bitmap file. To display a splash screen, you must specify a valid bitmap file.	Check the Knowledge Base for information about this error, or request technical support.
-6606	This setup has the same package code as the setup specified by minor upgrade item '%1'. The package codes must be unique.	Check the Knowledge Base for information about this error, or request technical support.
-6605	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6604	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6603	An error occurred while attempting to open the upgraded image %1.	Check the Knowledge Base for information about this error, or request technical support.
-6602	An error occurred while attempting to close the upgraded image %1.	Check the Knowledge Base for information about this error, or request technical support.
-6601	An error occurred while attempting to set the property %1 in the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6600	An error occurred while attempting to retrieve the property %1 from the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6599	An error occurred while creating the standard QuickPatch actions to the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6598	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6597	An error occurred while attempting add ISQuickPatchHelper.dll to the binary table in the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6596	An error occurred while attempting to sequence the action %1 in the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6595	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6594	An error occurred while attempting to determine the sequence number for the base action %1 in the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6593	An error occurred while initializing file class wrapper for %1.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6592	An error occurred while attempting to determine how to process the file %1.	Check the Knowledge Base for information about this error, or request technical support.
-6591	An error occurred while attempting to make the settings required to patch the file %1.	Check the Knowledge Base for information about this error, or request technical support.
-6590	An error occurred while attempting to create the RemoveFile table entry for %1.	Check the Knowledge Base for information about this error, or request technical support.
-6589	An error occurred while attempting to attach the component '%1' to its designated features.	Check the Knowledge Base for information about this error, or request technical support.
-6588	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6587	An error occurred while generating a formatted file name for %1.	Check the Knowledge Base for information about this error, or request technical support.
-6586	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6585	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6584	An error occurred while attempting to create a new component.	Check the Knowledge Base for information about this error, or request technical support.
-6583	An error occurred while attempting to load the component %1.	Check the Knowledge Base for information about this error, or request technical support.
-6582	An error occurred while saving the component %1.	Check the Knowledge Base for information about this error, or request technical support.
-6581	An error occurred while adding component '%1' to feature %2.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6580	An error occurred while generating a sub feature for feature %1.	Check the Knowledge Base for information about this error, or request technical support.
-6579	An error occurred while synchronizing data in table '%1' for component %2.	Check the Knowledge Base for information about this error, or request technical support.
-6578	An error occurred while synchronizing shortcuts from component '%1' while creating the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6577	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6576	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6575	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6574	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6573	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6572	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6571	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6570	An error occurred while writing the property '%1' to the patch configuration properties file.	Check the Knowledge Base for information about this error, or request technical support.
-6569	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6568	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6567	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6566	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6565	An error occurred while writing the image family '%1' to the patch configuration properties file.	Check the Knowledge Base for information about this error, or request technical support.
-6564	An error occurred while deleting the target image '%1' from the patch configuration properties file.	Check the Knowledge Base for information about this error, or request technical support.
-6563	An error occurred while writing the target image '%1' to the patch configuration properties file.	Check the Knowledge Base for information about this error, or request technical support.
-6562	An error occurred while writing the upgraded image %1 to the patch configuration properties file.	Check the Knowledge Base for information about this error, or request technical support.
-6561	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6560	The stored package code of %1, does not match %2 which is in the referenced package %3. Modifying a referenced MSI package of a Quick Patch, invalidates all Quick Patches that reference that MSI package.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6558	The Custom Action %1 in the InstallExecuteSequence table is run from an installed file. To run the custom action successfully, you may need to have a condition that checks if the source file is installed locally.	To successfully run a custom action that is run from an installed file, you must ensure that the file is installed locally using a conditional statement: • If the custom action is sequenced before RemoveFiles—Run the action only if the component was installed locally. (?ComponentName=3) • If the custom action is sequenced between RemoveFiles and InstallFiles—Run the action only if the component was installed locally. Do not run the action on an uninstallation. (?ComponentName=3) AND NOT($ComponentName=2) • If the custom action is sequenced after InstallFiles—Run the action only if the component will be installed locally. ($ComponentName=3)  ***Note*** • *The ComponentName is the component that is associated with the source file.*
-6557	Unable to find the .NET Compact Framework for the given platform/processor combination from the **InstallShield Program File Folder\Support\NetCF.ini** file.	Ensure that the .NET Compact Framework is in the correct location.
-6556	Error in including .NET Compact Framework .cab files in setup.	This error might occur for the following reasons: • The .NET Compact Framework files cannot be found in the locations specified in the InstallShield **Program Files Folder\Support\.NetCR.ini** file. • The CEDefault.ico file is not in the InstallShield **Program File Folder\Program\04xx** folder. • There is an error writing to the **ISCEInstall** table.
-6555	Unable to copy **InstallShield Program File Folder\Support\CFAppMgr.ini** file to a temporary location.	Make sure that **CFAppMgr.ini** file exists at the InstallShield **Program Files Folder\Support** folder. If this file does not exist, launch the InstallShield setup in repair mode. Remove any temporary files to create more disk space on the system.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6553	Unable to find the .NET Compact Framework .cab file for the processor.	Open the InstallShield **Program Files Folder\Support\NetCF.ini** file and configure the .NET CF selection for this processor to resolve the issue.
-6552	The file key %1 of component %2 could not be synchronized with the previous package because that file key is already in use.	Check the Knowledge Base for information about this error, or request technical support.
-6551	The Name and Value columns are blank in the registry record Registry1 of Component NewComponent1. The Windows Installer will set the (Default) value of *HKEY_LOCAL_MACHINE\New Key* #1 to an empty string. If you would like to set the (Default) value to (Value Not Set), you need to set "Install if absent, Uninstall if present" for that key in the Registry view. If the key already exists on the Target machine, the (Default) value will not be changed.	This warning appears for every registry key you add to your project and do not manually set the default key. To set the (Default) value to (Value Not Set), you need to set "Install if absent, Uninstall if present" for that key in the Registry view.
-6550	%1 is not a valid .NET assembly to be installed to the Global Assembly Cache (where %1 is replaced with a full file path).	Only valid assemblies can be installed to the Global Assembly Cache. Ensure that the file selected for installation is a valid assembly.
-6549	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6548	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6547	The file '%1' already exists in the setup. Please modify the existing file instead of inserting a new one otherwise this file may not uninstall properly.	Check the Knowledge Base for information about this error, or request technical support.
-6546	An error occurred while testing the new file '%1' for potential conflicts.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6545	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6544	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6543	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6542	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6541	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6540	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6539	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6538	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6537	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6536	An error occurred opening %1.	Check the Knowledge Base for information about this error, or request technical support.
-6535	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6534	An error occurred cloning the component %1.	Check the Knowledge Base for information about this error, or request technical support.
-6533	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6532	An error occurred setting the updated component name for file %1.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6530	Error processing setup [1] for Upgrade Item [2]. This does not appear to be a valid setup.	This error occurs when a minor upgrade entry exists in the Upgrades view of InstallShield, and the previous installation specified does not exist in the location indicated. See Knowledge Base article Q108525 for resolution information.
-6529	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6528	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6528	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6527	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6526	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6525	The custom action %1 in the InstallExecuteSequence table is run from an installed file. It must be sequenced after the %2 action. Ensure that the custom action is sequenced properly and that the base action exists in the sequence.	This error occurs when you have authored a custom action that runs from a file that is installed during your setup. The custom action must come after the InstallFiles action in the sequence. Otherwise the file will not exist on the target system when Windows Installer runs the custom action. The only exception to this is when you run this custom action on uninstallation. In that case, the file has already been installed so the custom action does not have to occur after InstallFiles.
-6524	The Custom Action %1 in the InstallExecuteSequence table is deferred and must be sequenced between %2 and %3. Ensure that the Custom Action is sequenced properly and that the base actions exist in the sequence.	Check the Knowledge Base for information about this error, or request technical support.
-6523	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6522	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6521	An error occurred setting the required attribute on the feature %1.	Check the Knowledge Base for information about this error, or request technical support.
-6520	An error occurred deleting the COM data for component %1.	Check the Knowledge Base for information about this error, or request technical support.
-6519	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6518	An error occurred deleting the file %1 from the SelfReg table.	Check the Knowledge Base for information about this error, or request technical support.
-6517	An error occurred adding the file %1 to the SelfReg table.	Check the Knowledge Base for information about this error, or request technical support.
-6506	The Msi Command Line Attribute of the release currently being built is setting the REINSTALL or REINSTALLMODE parameter. This will cause your setup to always run in Resume mode.	Check the Knowledge Base for information about this error, or request technical support.
-6504	The Latest Image %1 does not contain any Previous Images.	Check the Knowledge Base for information about this error, or request technical support.
-6501	An error occurred building the ISDLLWrapper table. Component %1 is not found in the Component table.	Check the Knowledge Base for information about this error, or request technical support.
-6500	An error occurred building the ISDLLWrapper table. The File %1 is not found in the File table.	Check the Knowledge Base for information about this error, or request technical support.
-6499	The Shallow folder structure setting should not be used with multi-disk releases.	You can configure the Shallow Folder Structure setting on the Build tab of the Releases view. If error -6499 appears, select No for the Shallow Folder Structure setting so it will create a regular build folder structure.
-6497	Patch Creation Warning: %1.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6496	The file hash for the unversioned file '%1' does not differ from that of the previous version. This file will not be updated.	Check the Knowledge Base for information about this error, or request technical support.
-6495	The version for file '%1' only differs past the third version element. It must differ within the first three elements of the version string.	Check the Knowledge Base for information about this error, or request technical support.
-6494	The new file for file '%1' has the same version as the file that exists in the setup. This file will not be updated.	Check the Knowledge Base for information about this error, or request technical support.
-6493	The new file for file '%1' has a lower version than the file that exists in the setup. This file will not be updated.	Check the Knowledge Base for information about this error, or request technical support.
-6492	The file '%1' has no associated feature selected. This setting is required for adding new files.	Check the Knowledge Base for information about this error, or request technical support.
-6491	The new file path '%1' does not exist for file '%2'.	Check the Knowledge Base for information about this error, or request technical support.
-6490	Unable to set Just-In-Time compilation at install time because the Component %1 is configured to install to the Global Assembly Cache.	Check the Knowledge Base for information about this error, or request technical support.
-6489	Unable to set Installer Class because the Component %1 is configured to install to the Global Assembly Cache.	Check the Knowledge Base for information about this error, or request technical support.
-6487	The latest image %1 does not contain any previous images.	This error occurs if a latest image in the Patch Design view does not have at least one previous image. A latest image must have at least one previous image to create a patch.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6486	The CAB files will not be digitally signed because the version of MSI.DLL is less than 2.0.	Check the Knowledge Base for information about this error, or request technical support.
-6485	The MsiFileHash table will not be built because the version of MSI.DLL is less than 2.0.	Check the Knowledge Base for information about this error, or request technical support.
-6584	The External File key '%1' does not exist in the Upgraded Image '%2'.	Check the Knowledge Base for information about this error, or request technical support.
-6483	The External File reference '%1' does not exist.	Check the Knowledge Base for information about this error, or request technical support.
-6482	An error occurred while verifying an External File key.	Check the Knowledge Base for information about this error, or request technical support.
-6481	The Target Image '%1' does not exist.	Check the Knowledge Base for information about this error, or request technical support.
-6480	The Upgraded Image '%1' does not exist.	Check the Knowledge Base for information about this error, or request technical support.
-6479	Unable to resolve target path for Directory Id '%1'.	Check the Knowledge Base for information about this error, or request technical support.
-6478	FON files must have a Font Title. Font %1 in component %2.	Check the Knowledge Base for information about this error, or request technical support.
-6477	An error occurred while adding the remove registry entry for the key '%1' and Value '%2'.	Check the Knowledge Base for information about this error, or request technical support.
-6476	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6475	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6474	An error occurred while processing the registry value modifications made to the project.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6473	An error occurred while preparing to delete the value '%1' from the key '%2'.	Check the Knowledge Base for information about this error, or request technical support.
-6472	An error occurred while deleting the value '%1' from key '%2' for all features.	Check the Knowledge Base for information about this error, or request technical support.
-6471	An error occurred while deleting the value '%1' from key '%2' for feature '%3'.	Check the Knowledge Base for information about this error, or request technical support.
-6470	An error occurred while processing the registry key modifications made to the project.	Check the Knowledge Base for information about this error, or request technical support.
-6469	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6468	An error occurred while making the settings to modify the registry key '%1' to '%2' for all features.	Check the Knowledge Base for information about this error, or request technical support.
-6467	An error occurred while making the settings to modify the registry key '%1' to '%2' for feature '%3'.	Check the Knowledge Base for information about this error, or request technical support.
-6466	Error in populating the CE setup info into the desktop setup.	For additional information regarding this error, see the log file created in the following directory: **<Product Config>\<Release>\CEApps\<CE Project Name>**
-6465	Error in creating the CE setup media from the .inf file.	For additional information regarding this error, see the log file created in the following directory: **<Product Config>\<Release>\CEApps\<CE Project Name>**
-6464	Error in adding CE Setup DLLs info while creating the .inf file to build CE setup media.	For additional information regarding this error, see the log file created in the following directory: **<Product Config>\<Release>\CEApps\<CE Project Name>**
-6463	Error in adding CE registry entries.	For additional information regarding this error, see the log file created in the following directory: **<Product Config>\<Release>\CEApps\<CE Project Name>**

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6462	Error in adding CE file associations.	For additional information regarding this error, see the log file created in the following directory: **<Product Config>\<Release>\CEApps\<CE Project Name>**
-6461	Error in adding CE shortcuts.	For additional information regarding this error, see the log file created in the following directory: **<Product Config>\<Release>\CEApps\<CE Project Name>**
-6460	Error in adding CE setup files.	For additional information regarding this error, see the log file created in the following directory: **<Product Config>\<Release>\CEApps\<CE Project Name>**
-6459	Error in populating the information for the .inf file to build CE Setup media.	For additional information regarding this error, see the log file created in the following directory: **<Product Config>\<Release>\CEApps\<CE Project Name>**
-6458	An error occurred while creating an Additional File reference for the upgraded image '%1'.	Check the Knowledge Base for information about this error, or request technical support.
-6457	'Ignore Missing Files' is set on the target image '%1'. This cannot be set when using the 'Include Whole Files' option on the Upgraded Image.	Check the Knowledge Base for information about this error, or request technical support.
-6456	An unexpected error occurred while making the settings required to include whole files in the patch.	Check the Knowledge Base for information about this error, or request technical support.
-6455	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6454	Unable to create whole file patch for '%1'. An error occurred while attempting to rename the file.	Check the Knowledge Base for information about this error, or request technical support.
-6453	An error occurred while trying to rename the file '%1' back to its original file name '%2'.	Check the Knowledge Base for information about this error, or request technical support.
-6452	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6451	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6449	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6448	An error occurred while attempting to remove the file '%1' from the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6447	An error occurred while deleting the file table entry for the file '%1'.	Check the Knowledge Base for information about this error, or request technical support.
-6446	An error occurred while deleting the component key for the upgraded image for the file '%1'.	Check the Knowledge Base for information about this error, or request technical support.
-6445	An error occurred while adding the file '%1' to the upgraded image.	Check the Knowledge Base for information about this error, or request technical support.
-6444	An error occurred while creating the new file entry for file key '%1'.	Check the Knowledge Base for information about this error, or request technical support.
-6443	An error occurred while creating a new component for the file key '%1'.	Check the Knowledge Base for information about this error, or request technical support.
-6442	Unable to create new feature off of the root feature '%1'.	Check the Knowledge Base for information about this error, or request technical support.
-6441	Unable to create feature component entry for feature '%1' and component '%2'.	Check the Knowledge Base for information about this error, or request technical support.
-6440	Unable to find feature '%s' while creating settings for new file.	Check the Knowledge Base for information about this error, or request technical support.
-6434	The setup '%1' specified by ISLatestRelease was built compressed. To use this variable you must build your setup uncompressed.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6433	An error occurred while setting the property %1.	Check the Knowledge Base for information about this error, or request technical support.
-6432	An error occurred while creating the configuration data Properties.	Check the Knowledge Base for information about this error, or request technical support.
-6431	An error occurred while creating configuration data for the Image Family %1.	Check the Knowledge Base for information about this error, or request technical support.
-6430	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6429	An error occurred while creating configuration data for the Target Image %1.	Check the Knowledge Base for information about this error, or request technical support.
-6428	An error occurred while creating configuration data for the Upgraded Image %1.	Check the Knowledge Base for information about this error, or request technical support.
-6427	The setup referred to by , '%1', does not exist.	Check the Knowledge Base for information about this error, or request technical support.
-6426	An error occurred while trying to resolve the variable .	Check the Knowledge Base for information about this error, or request technical support.
-6425	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6424	The project contains a Minor Upgrade item, however the current release does not use a setup launcher. You will have to specify the upgrade command line manually for this minor upgrade to function properly.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6423	The project contains a Minor Upgrade item that references the setup %1. However, the referenced setup can only be upgraded with a Major upgrade. Please use the Major Upgrade item or the automatic upgrade item.	Check the Knowledge Base for information about this error, or request technical support.
-6422	An error occurred while adding Major Upgrade settings for %1.	Check the Knowledge Base for information about this error, or request technical support.
-6421	An error occurred determining if the setup %1 is a major upgrade.	Check the Knowledge Base for information about this error, or request technical support.
-6420	The Upgrade Item setup [MSI name].msi is not found. Unable to create upgrade settings.	Ensure that the setup exists, that it is an .msi file, and that the name is correct.
-6419	An unexpected error occurred processing upgrade item %1.	Check the Knowledge Base for information about this error, or request technical support.
-6418	An unexpected error occurred while processing items to be upgraded.	Check the Knowledge Base for information about this error, or request technical support.
-6415	There was an error creating the patch package. Writing contents of log file '%1' to output window.	Check the Knowledge Base for information about this error, or request technical support.
-6414	An error occurred populating the file hash for the unversioned file %1.	Check the Knowledge Base for information about this error, or request technical support.
-6413	The patch creation process returned the error code %1.	Check the Knowledge Base for information about this error, or request technical support.
-6412	An error occurred while creating the patch package.	Check the Knowledge Base for information about this error, or request technical support.
-6411	An error occurred setting the file version for %1.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6410	An error occurred while attempting to copy the file %s to the release location.	Check the Knowledge Base for information about this error, or request technical support.
-6409	An error occurred while building the source path for file %1.	Check the Knowledge Base for information about this error, or request technical support.
-6408	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6407	An error occurred setting the PackageCode int he Updated Image.	Check the Knowledge Base for information about this error, or request technical support.
-6406	An error occurred setting the Product Version in the updated image.	Check the Knowledge Base for information about this error, or request technical support.
-6405	An error occurred while preparing the Upgraded Image.	Check the Knowledge Base for information about this error, or request technical support.
-6404	An error occurred preparing the Patch Configuration Properties file.	Check the Knowledge Base for information about this error, or request technical support.
-6403	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6402	An error occurred while preparing the release location for building.	Check the Knowledge Base for information about this error, or request technical support.
-6401	Unable to copy Original Setup file to release location.	Check the Knowledge Base for information about this error, or request technical support.
-6400	Unable to copy patch configuration properties template to release location.	Check the Knowledge Base for information about this error, or request technical support.
-6309	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6308	mscorsn.dll cannot be located on your system. You can set the path to mscorsn.dll using Tools | Options. The file is part of the Microsoft .NET Framework redistributable. %1.	Check the Knowledge Base for information about this error, or request technical support.
-6307	Unable to extract one or more files to '%1'. The file path is longer than the limit set by the operating system. Change the build location of the current release to a shorter path to resolve this issue.	Change the build location of the current release to a shorter path.
-6306	Port number specified for configuring IIS Virtual Directories is invalid.	Make sure that the TCP port number that is specified for the virtual directory in the Internet Information Services view is valid. To learn more, see Configuring the TCP Port and Site Numbers.
-6305	Could not load assembly "%1" because of : "%2".	Make sure that you are able to load the dependent file and that the assemblies are accessible according to the assembly binding rules. For more information, see “How the Runtime Locates Assemblies”, available in the .NET Framework SDK help or in MSDN.
-6304	Error validating with CUB file "%1".	Ensure that the CUB file is valid.
-6303	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-6302	InstallShield could not resolve Visual Studio .NET project output "%1" from component "%2". You need to build the project in Visual Studio .NET or by using the Visual Studio .NET command line: devenv /build ConfigName [/project ProjName] [/projectconfig ConfigName] SolutionName	Build the project in Microsoft Visual Studio or by using the Visual Studio command line.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6274	Setup.exe is not found in the Disk1 location: %1. Please make sure that the previous full build of your release was completed if you are running the Build Tables Only.	This error occurs when the setup attempts to get the stamp from Setup.exe, but Setup.exe does not exist in the Disk1 folder. This happens when you cancel a full build in the middle of the process, and then run the Build Tables Only option.
-6273	A VB Script custom action exists in the project, but the VBScriptRuntime Merge Module has not been included.	Use the Redistributables view to add the VBScriptRuntime merge module to your project.
-6271	File %1 not found. An error occurred building the MsiFileHash table record for this file. Verify that the file exists in the specified location.	Ensure that the identified file exists in the specified location. If the file includes the predefined path folder VSSolutionFolder, see Using the VSSolutionFolder Path Variable with Visual Studio Solutions.
-6270	The record %1 in the Icon table exceeds the limit of %2 characters. As a result, the build will be unable to persist the database.	Make sure that the record is at or under the limited number of characters.
-6269	An error occurred copying directory %1 to %2. Please ensure that the source directory's path is correct.	Check the source directory's path.
-6268	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-6267	An error occurred while extracting files from the cab file %1 to the location %2	Ensure that the location is accurate.
-6266	An error occurred while attempting to create and initialize the CAB extraction engine for %1	This error can occur when one or more files is not registered. Ensure that all files are registered.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6265	An error occurred building feature '%1'. This feature has the same name (with different case) as another feature in the project. Duplicate feature names are not allowed in InstallScript MSI projects. Rename one of the features to fix this error.	Remove or rename one of the features.
-6264	A record in the %1 table is using string ID '%2' for column '%3' but this string is blank and the column is not nullable.	Provide a string for use in the column.
-6263	The number of allowed InstallScript custom actions (1000) has been exceeded. Please remove some InstallScript custom actions and rebuild the setup.	This error appears when a Basic MSI, DIM, InstallScript MSI, or Merge Module project has more than 1,000 InstallScript custom actions. Use the Custom Actions and Sequences view (or the Custom Actions view) to remove InstallScript custom actions so that your project no longer exceeds the maximum number allowed.
-6262	The Directory table contains the entry %1. This identifier is reserved. Defining it in the Directory table will yield unpredictable results at run time. Use the Direct Editor to rename this directory identifier.	Go to the Directory table in the Direct Editor and rename the directory identifier.
-6260	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-6259	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-6258	An error occurred extracting digital signature information from file "%1". Make sure the digital signature information provided in the IDE is correct.	Ensure that the digital signature information is correct. You can access the digital signature information in the Releases view or in the Release Wizard's Digital Signature panel.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6257	The build engine has detected that PRODUCT_NAME has been defined in the string table. At run time, your setup will display this value as your Product Name instead of the Product Name values defined in the IDE.	This warning message occurs when the ID PRODUCT_NAME has been defined in the String Editor view. This string identifier overrides the Product Name values set in the General Information view or for the product configuration in the Releases view.
-6255	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-6254	An error occurred building the MsiFileHash table for File %1	Check the Knowledge Base for information about this error, or request technical support.
-6253	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-6252	Internal build error	Check the Knowledge Base for information about this error, or request technical support.
-6251	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6250	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6249	Component %1 is configured to install to the Global Assembly Cache but includes one or more subdirectories.	This error occurs when you are using a .NET Project Output that includes subdirectories. InstallShield does not currently support installing these files to the Global Assembly Cache. To avoid this error, change the directory for the component (for example, to INSTALLDIR), or statically add the files and configure them.
-6248	Could not find dependent file %1, or one of its dependencies of component %2	Use the "Build Tables & Refresh Files" option to build the release if the release location is in *<ISProjectDataFolder>* or *<ISProjectFolder>*. For more information, see "How the Run time Locates Assemblies", available in the .NET Framework SDK help or on MSDN.
-6247	The .NET Framework redistributable %1 is not found on the system. %2	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6246	Could not add dotnetfx.exe to the built image. If you are including .NET 1.1, make sure a core language is selected.	Check the Knowledge Base for information about this error, or request technical support.
-6245	One or more of the project's components contain .NET properties that require the .NET Framework. It is recommended that the release include the .NET Framework.	Consider adding the .NET Framework redistributable to your project. If the .NET Framework is not present on the target system at run time, the installation installs it. To learn more, see Adding .NET Framework Redistributables to Projects.
-6244	One or more of the project's components contain .NET properties that require the .NET Framework. However, the .NET Framework cannot be detected. %2	Check the Knowledge Base for information about this error, or request technical support.
-6243	InstallUtilLib.dll cannot be located on your system. This file is required for Installer custom actions. You can set the path to InstallUtilLib.dll using Tools | Options. The file is part of the Microsoft .NET Framework redistributable. %1	Set the path to InstallUtilLib.dll by selecting Options from the Tools menu and clicking the .NET tab.
-6242	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6241	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6240	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6239	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6238	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6237	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6236	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6235	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6234	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6233	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6232	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6231	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6230	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6229	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6228	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6227	.NET scan of Component %1 failed	Check the Knowledge Base for information about this error, or request technical support.
-6226	Opening Visual Studio .NET solution	Make sure that the solution exists at the location specified. To find the solution information, navigate to the InstallShield table in the Direct Editor and find the value for ISDotNetSolution. In addition, ensure that Visual Studio is installed on the build machine.
-6225	Resolving Visual Studio .NET project output "%1"	Make sure the project exists in the solution. To find the solution information, navigate to the InstallShield table in the Direct Editor and find the value for ISDotNetSolution. In addition, ensure that project has the specified output.
-6224	Processing merge module %1 of feature %2	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6223	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6222	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6221	Could not resolve Visual Studio .NET project output "%1" from component %2	Make sure the project exists in the solution. To find the solution information, navigate to the InstallShield table in the Direct Editor and find the value for ISDotNetSolution. In addition, ensure that project has the specified output.
-6220	Dynamically acquired %1 data conflicts with static data associated with component %2. Overwriting with dynamic data.	This error occurs when manual entries made to the MsiAssembly and MsiAssemblyName tables conflict with .NET assembly properties that result from scanning. To eliminate this error, set the component's .NET Scan at Build property to None or delete the values that were manually added to the MsiAssembly and MsiAssemblyName tables.
-6219	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6218	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6217	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6216	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6215	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6214	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6213	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6212	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6211	Destination of Component %1 is GlobalAssemblyCache but key file is not a .NET assembly	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6210	An error occurred building COM .NET Interop information for Component %1	InstallShield calls regasm.exe on the .NET .dll to create a .reg file. This should create a .reg file. Then, the .reg file is imported. If either of these steps fails, error -6210 occurs. Make sure that the component's key file is a .NET .dll and that the .NET assembly has "types to export".
-6209	Regasm.exe cannot be located on your system. This file is required to obtain .NET COM Interop information for Components. You can set the path to Regasm.exe using Tools | Options. The file is part of the Microsoft .NET Framework redistributable. %1	Set the path to Regasm.exe by selecting Options from the Tools menu and clicking the .NET tab.
-6208	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6207	An error occurred building Setup File %s.	Check the Knowledge Base for information about this error, or request technical support.
-6205	An error occurred building Component %1. The Component's destination directory %2 is not found in the directory table. Change the Component's destination to a valid location.	Check the Knowledge Base for information about this error, or request technical support.
-6204	An error occurred importing %1 for Component %2. Make sure the file exists in the specified location and that the file is a valid REG file.	Check the Knowledge Base for information about this error, or request technical support.
-6203	Internal build error.	Check the Knowledge Base for information about this error, or Check the Knowledge Base for information about this error, or request technical support.
-6202	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6201	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6200	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6199	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6197	An error occurred streaming %1 into %2. Check to make sure the .msi package is not currently in use by another process.	Ensure that the .msi package is not currently being used by another process.
-6196	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6195	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6194	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6193	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6192	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6191	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6190	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6189	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6188	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6187	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6186	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6185	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6184	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6183	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6182	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6181	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6180	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6179	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6178	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6177	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6176	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6175	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6174	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6173	An error occurred renaming file %1 to %2	Make sure that you have enough free hard disk space on the drive that contains the interim folder under your product configuration (for example, **\MySetups\Your Project Name-27\Product Configuration 1\Interm**).

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6172	An error occurred while creating a CAB file. For incremental compressed builds, make sure your project includes at least one file or rebuild your media.	Make sure that your project includes at least one file or rebuild your media.
-6171	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6170	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6169	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6168	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6167	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6166	An error occurred updating the Word Count Summary Property of the Summary Information Stream.	Check the Knowledge Base for information about this error, or request technical support.
-6165	An error occurred exporting table %1.	Check the Knowledge Base for information about this error, or request technical support.
-6164	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6163	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6162	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6161	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6160	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6159	The data type for a column in a table in the source project is incompatible with that of the target database.	Use the Direct Editor to ensure the data type is correct in the specified column and table.
-6158	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6157	Target field in table is too small for data.	Use the Direct Editor to ensure the data can fit in the allocated table field.
-6156	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6155	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6154	Build canceled by the user.	This error occurs only when the build is terminated during the build process.
-6153	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6152	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6151	Cannot save target database.	Check the Knowledge Base for information about this error, or request technical support.
-6150	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6149	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6148	Cannot insert record in the specified table.	Check the Knowledge Base for information about this error, or request technical support.
-6147	Cannot update the specified target field in the table.	Check the Knowledge Base for information about this error, or request technical support.
-6146	Cannot create new record in the specified table.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6145	Cannot retrieve value of the specified column in the table.	Check the Knowledge Base for information about this error, or request technical support.
-6144	Cannot open database view for table %1.	Check the Knowledge Base for information about this error, or request technical support.
-6143	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6142	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6141	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6140	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6139	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6138	Error compiling Setup.rul.	Setup.rul is required for the project. Check to ensure that this file is included in the project.
-6137	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6136	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6135	The specified feature is not associated with any Setup Type.	Associate the specified feature with a Setup Type.
-6134	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6133	An error occurred updating properties specified in the current Product Configuration.	Check the properties specified in the current Product Configuration to ensure they are valid.
-6132	Error Building the Storages table.	Use the Direct Editor to verify that the data in the table is correct.
-6131	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6130	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6129	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6128	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6127	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6126	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6125	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6124	Unknown Exception.	Check the Knowledge Base for information about this error, or request technical support.
-6123	Unknown Exception.	Check the Knowledge Base for information about this error, or request technical support.
-6122	No components are included in the build.	Ensure your build contains at least one component.
-6121	Error occurred copying built merge modules to the Modules folder.	Check the Knowledge Base for information about this error, or request technical support.
-6120	The Resource Linker failed to build the specified DLL.	Check the Knowledge Base for information about this error, or request technical support.
-6119	The resource linker could not be found.	Open the Options dialog, available from the Tools menu, and set a location to a Resource linker.
-6118	The Resource compiler failed to build the specified RES file required to link the DLL.	Check the Knowledge Base for information about this error, or request technical support.
-6117	The resource compiler could not be found.	Open the Options dialog, available from the Tools menu, and set a location to a Resource compiler.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6116	Failed to export .rc file from project: %1.	Check the Knowledge Base for information about this error, or request technical support.
-6115	Unable to retrieve version information from ISSetup.dll.	This error may occur if the **ISSetup.dll** file in the following location is corrupted, or if it has been removed from the build machine. This file should be in the following location: **InstallShield Program Files Folder\redist\Language Independent\i386** To resolve this error, try running a repair of InstallShield.
-6114	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6113	An error occurred during the incremental build.	Check to ensure that a previous media was built before selecting "Build Tables Only" or "Build Tables and Refresh Files" and ensure the previous media was not deleted.
-6112	Error deleting a file streamed into Setup.exe.	Check the Knowledge Base for information about this error, or request technical support.
-6111	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6110	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6109	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6108	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6107	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6106	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6105	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6104	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6103	Could not find file %1.	Check the file to ensure it exists in the specified location. If the file includes the predefined path folder VSSolutionFolder, see Using the VSSolutionFolder Path Variable with Visual Studio Solutions.
-6102	Error searching for dynamic files matching "%1".	Make sure the dynamic file flag specified is valid.
-6101	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6100	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6099	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6098	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6097	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6096	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6095	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6094	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6093	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6092	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6091	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6090	The scrollable text file:'%1' specified for the Control:'%2' does not exist.	Check to make sure the text file exists in the specified location %1 will contain the full path to an .rtf file. If it is empty, the ISBuildSourcePath column for control %2 is empty. For Scrollable text controls, ISBuildSourcePath in the Control table should point to a valid file.
-6089	Both the Text and ISBuildSourcePath columns for Control:'%1' contain path information, the Text column will be used.	The Control %1 cannot have entries in both ISBuildSourcePath and the Text columns of the Control table. Go to the Direct Editor and remove the unused information in either ISBuildSourcePath or Text column.
-6088	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6087	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6086	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6085	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6084	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6083	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6082	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6081	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6080	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6079	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6078	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6077	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6076	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6075	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6074	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6073	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6072	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6071	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6070	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6069	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6068	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6067	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6066	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6065	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6064	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6063	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6062	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6061	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6060	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6059	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6058	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6057	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6056	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6055	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6054	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6053	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6052	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6051	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6050	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6049	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6048	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6047	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6046	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6045	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6044	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6043	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6042	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6041	Internal build error.	This build error can occur if a project includes support for multiple languages and if the release is configured to remove unused directories from the **Directory** table of the .msi package. To resolve this error: **1.** In the View List under **Media**, click **Releases**. **2.** In the **Releases** explorer, click the release that you were building when you encountered this error. **3.** Click the **Build** tab. **4.** For the **Keep Unused Directories** setting, select **Yes**. For more information, see Build Tab for a Release.
-6040	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6039	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6038	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6037	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6036	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6035	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6034	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6033	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6032	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6031	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6030	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6029	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6028	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6027	InstallShield cannot convert the value of the ISBuildSourcePath column in the File table for the Visual Studio .NET output files. InstallShield is not running within Visual Studio .NET. *OR* InstallShield cannot convert the value of the ISBuildSourcePath column in the File table for the Visual Studio .NET output files. The opened Visual Studio .NET solution name is different from the solution name stored in the .ism project.	Run the upgrade from within Microsoft Visual Studio to avoid this warning. *OR* Open the .ism project within the same solution when creating it in InstallShield.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6026	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6025	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6024	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6023	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6022	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6021	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6020	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6019	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6018	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6017	The build was unable to extract COM information; make sure that you are running as Administrator.	Extracting COM information in InstallShield requires administrative privileges. For more information, see Launching InstallShield with vs. Without Administrative Privileges.
-6016	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6015	Error generating short file name for file: %1.	Check the Knowledge Base for information about this error, or request technical support.
-6014	The shortcut for a component is invalid because it does not reference an icon resource.	Specify an icon file and icon index for this shortcut to resolve this issue.
-6013	The component condition for the specified component is invalid.	Use the Components view to modify the condition property of the specified component.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-6012	Failed to set codepage for language %1.	Install the codepage for the specified language.
-6011	Failed to open string table.	Check the Knowledge Base for information about this error, or request technical support.
-6009	An error occurred creating the Web media for Internet Explorer.	Check the Knowledge Base for information about this error, or request technical support.
-6008	An error occurred creating the Web media for Internet Explorer.	Check the Knowledge Base for information about this error, or request technical support.
-6007	Error occurred copying user specified uncompressed support files to Disk1.	Check the Knowledge Base for information about this error, or request technical support.
-6006	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-6005	An error occurred creating the Package Definition File.	Check the Knowledge Base for information about this error, or request technical support.
-6004	An error occurred streaming the digital certificate for Msi engine(s) into setup.exe.	Check the Knowledge Base for information about this error, or request technical support.
-6003	An error occurred streaming '%1' into setup.exe.	Check the Knowledge Base for information about this error, or request technical support.
-6002	Error while attempting to run the custom build setup for objects.	Check the Knowledge Base for information about this error, or request technical support.
-6001	An Error occurred while preprocessing ISObjectProperty. Make sure that ll items have a valid IncludeInBuild tag.	Check the Knowledge Base for information about this error, or request technical support.
-6000	An Error occurred gathering information about supplemental merge modules.	Check the Knowledge Base for information about this error, or request technical support.
-5099	An error occurred creating the build report.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-5098	The Directory table contains a circular reference.	Check the Knowledge Base for information about this error, or request technical support.
-5097	Could not write custom records to the _Validation table.	Check the Knowledge Base for information about this error, or request technical support.
-5096	Could not retrieve the standard table list.	Check the Knowledge Base for information about this error, or request technical support.
-5095	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5094	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5093	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5092	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5091	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5090	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5089	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5088	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5087	Stop at first error.	The user chose the "Stop build process when first error is encountered" in Tools | Options menu. An error occurred so the build was stopped.
-5086	Treat warnings as errors.	Check the Knowledge Base for information about this error, or request technical support.
-5085	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5084	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-5083	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5082	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5081	Could not write to setup.ini.	Check the Knowledge Base for information about this error, or request technical support.
-5080	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5079	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5078	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5077	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5076	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5075	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5074	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5073	The binary key %1 is missing.	Open the **Binary** table in the Direct Editor to verify that this entry exists.
-5072	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5071	Standard DLL Function syntax error.	Make sure you specify **void** if your function does not return anything. Click the ellipsis button (...) in the Function Signature field of the Custom Action to validate your signature.
-5070	Standard DLL Function Name syntax error.	Click the ellipsis button (...) in the Function Signature field of the Custom Action to validate your signature.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-5069	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5068	Standard DLL Function parameters syntax error.	Click the ellipsis button (...) in the Function Signature field of the Custom Action to validate your signature
-5066	Standard DLL Function return value syntax error.	Click the ellipsis button (...) in the Function Signature field of the Custom Action to validate your signature
-5065	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5064	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5063	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5062	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5061	The specified filename already exists.	Use the component's **Source Location** property to prevent this warning.
-5060	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5059	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5058	Could not obtain merge module dependencies.	Check the Knowledge Base for information about this error, or request technical support.
-5057	Could not obtain merge module catalog.	Check the Knowledge Base for information about this error, or request technical support.
-5056	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5054	Could not determine the size of the file "%1".	Verify the file specified exists.
-5053	The file "%1" could not be found.	Verify the file specified exists.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-5052	Could not determine the free space on the volume %1.	Check the Knowledge Base for information about this error, or request technical support.
-5051	Could not remove the read-only attribute from the file "%1".	Check the Knowledge Base for information about this error, or request technical support.
-5050	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5049	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5048	Could not create the file "%1".	Check the Knowledge Base for information about this error, or request technical support.
-5047	Cannot create directory %1.	Check the Knowledge Base for information about this error, or request technical support.
-5046	Could not preserve previous Build Reports.	This build error occurs when a release exists, but the corresponding Report Files folder in the release folder does not exist. This scenario typically occurs if you manually delete the Report Files folder, but not the disk image folder. This scenario may also occur if you upgrade a project from an earlier version of InstallShield. To resolve this error, click the Open Release Folder button. InstallShield opens the current release's folder. Delete all of the files within the folder. Then rebuild the release.
-5045	Could not preserve previous Build Logs.	This build error occurs when a release exists, but the corresponding Log Files folder in the release folder does not exist. This scenario typically occurs if you manually delete the Log Files folder, but not the disk image folder. This scenario may also occur if you upgrade a project from an earlier version of InstallShield. To resolve this error, click the Open Release Folder button. InstallShield opens the current release's folder. Delete all of the files within the folder. Then rebuild the release.
-5044	Cannot delete directory %1.	Check the Knowledge Base for information about this error, or request technical support.
-5043	The volume does not exist.	Check in the Releases view to ensure the volume specified exists.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-5042	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5041	The string ID "%1" was used to specify a Component or Feature destination.	You cannot use a stringID for a destination. Find the component or feature using this string ID and change the destination using a directory identifier.
-5040	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5039	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5038	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5037	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5036	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5033	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5032	Error building ListView table.	Use the Direct Editor to verify that the data in the table is correct.
-5031	Error building ListBox table.	Use the Direct Editor to verify that the data in the table is correct.
-5030	Error building ComboBox table.	Use the Direct Editor to verify that the data in the table is correct.
-5029	Error building CheckBox table.	Use the Direct Editor to verify that the data in the table is correct.
-5028	Error building RadioButton table.	Use the Direct Editor to verify that the data in the table is correct.
-5027	Error building Control table.	Use the Direct Editor to verify that the data in the table is correct.
-5026	Error building Dialog table.	Use the Direct Editor to verify that the data in the table is correct.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-5025	Could not save package.	Check the Knowledge Base for information about this error, or request technical support.
-5024	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5023	Error building File table.	Use the Direct Editor to verify that the data in the table is correct.
-5022	Error building Feature table.	Use the Direct Editor to verify that the data in the table is correct.
-5021	Error building Component table.	Use the Direct Editor to verify that the data in the table is correct.
-5020	Error building Directory table.	Use the Direct Editor to verify that the data in the table is correct.
-5019	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5018	The logical disk does not contain any features.	Check the Knowledge Base for information about this error, or request technical support.
-5017	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5016	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5015	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5014	An error occurred building icon "%1". The specified Icon key was not found in the Icon table.	See Knowledge Base article Q107116 for information about this error.
-5013	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5012	Invalid Feature_ value for %1.	Use the Direct Editor to open the specified table and ensure data in that table is valid.
-5011	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-5010	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5009	The schema Summary Stream must be 200 or later.	Check the Knowledge Base for information about this error, or request technical support.
-5008	Intel64 must be specified in the template of the Summary Stream.	Check the Knowledge Base for information about this error, or request technical support.
-5007	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5006	Could not save "%1".	Check the Knowledge Base for information about this error, or request technical support.
-5005	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5004	Could not open the project with write access.	Check to make sure the project is not already open by the IDE.
-5003	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5002	Internal build error.	Check the Knowledge Base for information about this error, or request technical support.
-5001	Could not copy setup.ini.	Check the Knowledge Base for information about this error, or request technical support.
-4701	An error occurred building the One-Click Install Web page.	Check the Knowledge Base for information about this error, or request technical support.
-4371	There were warnings compiling InstallScript.	Use the InstallScript Debugger to step through your InstallScript to identify problems.
-4370	There were errors compiling InstallScript.	Use the InstallScript Debugger to step through your InstallScript to identify problems.
-4354	The build was unable to extract COM information from a file in a component.	Ensure that the file is self-registering and verify that the self-registration process does not fail.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-4352	There is no key file for the specified component. A key file is required for dynamic COM extraction.	Specify the key file for the specified component from the Components view.
-4350	Dynamically acquired MIME text/ scriptlet conflicts with static data associated with component %1. Overwriting with dynamic data.	This warning occurs when you add a COM server .dll file using the Component Wizard, which does static COM extraction, and then later change the component's COM Extract at Build setting to Yes. The dynamically extracted data conflicts with the static data in the .ism file. Ensure that the COM extraction setting is accurate.
-4349	Failed to build %1 information for dynamically extracted COM components.	Check the Knowledge Base for information about this error, or request technical support.
-4348	Failed to build %1 information for dynamically extracted COM components.	Check the Knowledge Base for information about this error, or request technical support.
-4347	Failed to build information for dynamically extracted COM components.	Check the Knowledge Base for information about this error, or request technical support.
-4346	Failed to build information for dynamically extracted COM components.	Check the Knowledge Base for information about this error, or request technical support.
-4345	Failed to build information for dynamically extracted COM components.	Check the Knowledge Base for information about this error, or request technical support.
-4344	Failed to build information for dynamically extracted COM components.	Check the Knowledge Base for information about this error, or request technical support.
-4343	Failed to build information for dynamically extracted COM components.	Check the Knowledge Base for information about this error, or request technical support.
-4340	Failed to initialize COM extraction module.	Check the Knowledge Base for information about this error, or request technical support.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-4327	Could not write string ID "%1" to the InstallScript strings file.	This error pertains to projects containing InstallScript (either an InstallScript MSI project or a Basic MSI project with InstallScript custom actions). At build time, InstallShield creates an .ini file that contains all of the string entries in the project. In the error message, "%1" represents the line number in this .ini file that fails to get written. If the line number is 0, a general failure to create or write to the .ini file occurred. This might indicate that you are out of disk space.
-4303	An unexpected error occurred while synchronizing the file key in the specified component.	Check the Knowledge Base for information about this error, or request technical support.
-4302	A conflict was encountered while synchronizing file key %1 in component %2.	Check the Knowledge Base for information about this error, or request technical support.
-4301	Could not find the .msi file for key synchronization.	Check to ensure the specified MSI file exists.
-4093	Cannot copy source %1 to target %2	Ensure that the file exists, that there is enough disk space on the target location, and that you are able to copy to the location.
-4092	Error opening MSI database %1.	Check the Knowledge Base for information about this error, or request technical support.
-4075	File not found. An error occurred merging module '%1' for feature '%2'.	Verify that the merge module exists in the merge module search path. This error may occur if an installation project that contained an InstallScript MSI object was upgrade to InstallShield 2016, which no longer supports InstallScript MSI objects. If this is the case, remove that objects from your project through the Redistributables view. You can upgrade the InstallScript MSI object to InstallShield 2016, which will convert it to a merge module project. Then you can add that merge module to your installation project.
-4072	Error retrieving the dependencies for %1.	Check the Knowledge Base for information about this error, or request technical support.
-4006	Cannot delete file.	Ensure the specified file exists and is not set to read-only.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-3876	Ignoring invalid template %1 in the Summary Stream.	InstallShield generates this warning if the Template Summary setting contains an invalid entry. To learn about how to enter valid values, see Using the Template Summary Property.
-3851	Error building the specified table.	Use the Direct Editor to open the specified table and ensure data in that table is valid.
-3713	The function block must be in the form "Module::Function".	Change the function block to follow the form "Module::Function".
-3204	Cannot extract icon with specified index from the file specified for the icon.	Ensure the file containing the icon exists, and the index number is present in the file.
-3138	The Patch Configuration specified does not exist in the project. Please verify that the Patch Configuration exists and that the spelling and character case match.	When using the BuildPatchConfiguration method on the IPWiProject object, this error might occur.
-3114	Application path %1 does not contain destination of associated component %2.	This error occurs when an application path entry was created in the Advanced Settings section of the Components view, but no application path was specified. To correct this, set the application path entry. Because an application path is also a registry entry, this error might occur if a registry path like the following one is created, but it is missing the path entry: HKEY_LOCAL_MACHINE\Microsoft\Windows\CurrentVersion\App Paths To correct this, specify a path entry.
-3028	An error occurred replacing string IDs in the specified table.	A required string ID was left blank in the specified table. Provide a string ID.
-3016	Failed to add Binary table %1 to package.	Check the Knowledge Base for information about this error, or request technical support.
-2200	Could not overwrite file.	Ensure the specified file is not read-only.
-1531	The size specified for the disk is too small for the feature.	Using the Release Wizard, change the Media Format to a larger size.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-1530	The size specified for the disk is too small for the file.	Using the Release Wizard, change the Media Format to a larger size.
-1527	No files are included in the project.	This warning occurs when you have not included any files in your project. Use the Files and Folders view to add files into your project to avoid this error.
-1505	Could not add the CAB file to the MSI package.	Check the Knowledge Base for information about this error, or request technical support.
-1501	Could not compress "%1" into "%2".	Ensure that IsCmdBld.exe is executed in the relative directory. You can do this by locating the files specified in the error message and launching IsCmdBld.exe within their directory.
-1119	The specified CUB file does not exist.	Ensure that the CUB file name is spelled correctly.
-1027	Failed signing %1.	Check the digital signature information (digital keys and password) provided for the current release.
-1024	File not found. Cannot stream the file into the Binary table.	Ensure the specified file exists.
-1015	Cannot copy source '%1' to target %2.	Check the Knowledge Base for information about this error, or request technical support.
-1014	Cannot rename a directory.	Windows Explorer or a DOS prompt may be pointing to a subfolder of the release output folder (Disk1) or to the Interm folder, locking it. Change the current directory. Close any open files in the Disk1 folder. Close Msidb.exe if it is open.
-1013	The specified file is being used by another program.	Close the application that is currently using the file and re-run the build process.
-1009	Insufficient disk space or the target drive cannot be located.	Increase disk space on the build target, or select a new target for the build. If the target drive cannot be located, select a new target, or ensure permissions are set correctly for the target drive.
-1007	Cannot copy source %1 to target %2	Ensure that the file exists, that there is enough disk space on the target location, and that you are able to copy to the location.

Table 12-2 • Build Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Information
-1005	Could not compress %1 into %2 (where %1 is the full path to a file and %2 is the name of a .cab file).	This error could occur when you are building a compressed setup and run out of disk space or temporary space on the machine. Another reason that this error might occur is due to an internal error occurred with the CAB APIs. For more information, request technical support.
-1001	Error opening MSI database.	Check the Knowledge Base for information about this error, or request technical support.
-1000	Invalid product configuration. Failed to create directory.	Check the Knowledge Base for information about this error, or request technical support.

Media Build Errors and Warnings

Media build errors that occur before the build is generated are displayed in an error message dialog box; those that occur while the build is generated are displayed in the Output window and are recorded in the log file.

Table 12-3 • Media Build Errors and Warnings

Error or Warning Number	Message
118	Could not find CAB file with file #number. The system cannot find the path specified.
129	Media too small. Could not fit filename on disk number.
-5000	File group 'file group name' contains a link to a non-existent file - 'filename'.
-5006	'EnterPassword' dialog resource not found in resource library filename.
-5020	The built media files could not be uploaded to the site URL. The URL could not be parsed.
-5021	The built media files could not be copied to the folder 'path'.
-5032	Failed to sign the self-extracting executable file.
-5050	The font data could not be read from the project file. The 'Font' table may be missing from the project.
-5051	Unable to create font data file: 'data file path and name'
-7040	No features included in the media.

Table 12-3 • Media Build Errors and Warnings (cont.)

Error or Warning Number	Message
-7041	Could not create instance of CABEngine. CABEngine component might not be installed or registered properly. You may need to reinstall InstallShield.
-7043	Could not open the following media for the differential build - "path to media header file"
-7044	Product Version is not specified in the **Project Settings** property sheet.
-7045	File could not be found - "InstallShield **Program Files Folder\Redistributable\Compressed Files\Language Independent\OS Independent\Corecomp.ini**"
-7126	Not enough free disk space.
-7127	The Disk Images directory is use by another program.
-7275	Certificate invalid.
-7276	Certificate expired.
-7380	Build canceled by the user.
-9008	Failed saving object property.

Error 118

Message

`Could not find CAB file with file #number. The system cannot find the path specified.`

Troubleshooting Information

This error occurs when you build an already built media and one of the cabinet files that was previously created no longer exists.

If you encounter this error, rebuild the media.

Error 129

Message

`Media too small. Could not fit filename on disk number.`

Troubleshooting Information

This error can occur for any of the following reasons:

- Total size of files required to be on one disk is greater than selected disk size.
- An installation requires that certain files reside on the first disk, including the files placed in the Support Files view (such as Setup.bmp and License.txt) and Data1.cab. Also, an installation requires that certain files be compressed in Data1.cab, including the following:
  - files needed by the objects you have included in your setup
  - resource files for each language your installation supports
  - the file for the dialog box skin you selected

If your installation includes a large skin, setup files, or objects—or a large number of support files, objects, or languages—the total size of files required to be on the first disk can be greater than the disk size you selected in the Release Wizard's Media Type panel.

To avoid this error and complete the build successfully, you must either remove the skin or some support files or objects from your project, reduce the size of some support files or choose a skin with a smaller file, or select a larger disk size.

If the error message specifies a disk other than disk 1, some of the files in your components may be too large to fit on the disk even when they have been compressed. In this case, you must either remove the large files, reduce their size, or select a larger disk size.

- There is not enough space on your drives.

To create the installation's cabinet files, the release builder requires space in the system's temporary folder and in the build location. To avoid this error, make sure you have enough free space on the drive on which the system's temporary folder resides and the drive on which you have specified the build location.

You can find the system's temporary folder by typing in the word "set" at a DOS prompt. The temporary folder is identified by the value of the environment variable TEMP. You can find the build location by looking at the Release Wizard's General Options panel's Advanced dialog box or the Release view's Release Location property.

- There is not enough virtual memory.

On Windows NT, this error can be caused by a lack of free virtual memory rather than a lack of free hard disk space. For information, visit Knowledge Base article Q102757.

Warning -5000

Message

`Component 'component name' contains a link to a non-existent file - 'filename'`

Troubleshooting Information

The indicated component contains a link to the indicated file, which cannot be found. Remove the file from the component, or if you have renamed or moved the file, restore it to the indicated name and location.

Warning -5006

Message

`'EnterPassword' dialog resource not found in resource library filename.`

Troubleshooting Information

You have selected the Show Password dialog during setup initialization option in the Release Wizard's Password panel or the Show Password Dialog property in the Releases view, but one or more copies of **_Isres.dll** do not contain a resource template for this dialog box. (A separate instance of this warning is displayed for each copy of **_Isres.dll** that does not contain the resource template.) To fix this problem, you must use the Premier edition of InstallShield.

Warning -5020

Message

`The built media files could not be uploaded to the site URL. The URL could not be parsed.`

Troubleshooting Information

Verify that the URL you specified in the Release view's FTP Host Address property exists.

Warning -5021

Message

`The built media files could not be copied to the folder 'path'.`

Troubleshooting Information

You have selected the Copy the built media files to the folder option in the Release Wizard's Postbuild Options panel or the Copy to Folder property in the Release view; the built media files could not be copied to the folder that you specified with that option. Make sure that the path you specified exists and is writable.

Warning -5032

Message

`Failed to sign the self-extracting executable file.`

Troubleshooting Information

The build failed to sign the self-extracting executable file as specified in the Release Wizard's Digital Signature panel or the Release view's Sign Media property. Among the possible causes of this error are an invalid or missing private key file.

Warning -5050

Message

```
The font data could not be read from the project file. The 'Font' table may be missing from the project.
```

Troubleshooting Information

Check the Direct Editor to see if a Font table is included in your project. If it is not, and you cannot use source control software to roll back to a version of your project that did include that table, try deleting the font files from your project and then re-including them.

Warning -5051

Message

Unable to create font data file: '*data file path and name*'

Troubleshooting Information

Make sure that the specified path exists and is writable, and that a read-only file of the same name does not already exist at the same location.

Error -7040

Message

```
No features included in the media.
```

Description

Your specified media does not contain any features. All media builds must include at least one feature.

Troubleshooting Information

Make sure that you have created at least one feature in the Features view.

Error -7041

Message

```
Could not create instance of CABEngine. CABEngine component might not be installed or registered
    properly. You may need to reinstall InstallShield.
```

Troubleshooting Information

To resolve the problem, manually register the file MediaBuild40.dll (in the **InstallShield\MediaBuild** folder of your build machine's common files folder) using Regsvr32.exe (in the Windows system folder), using a DOS command line like the following:

```
Regsvr32.exe "C:\Program Files\Common Files\InstallShield\MediaBuild\MediaBuild40.dll"
```

Error -7043

Message

`Could not open the following media for the differential build - "path to media header file"`

Troubleshooting Information

The message refers to a media header file that you specified in the Differential Media list box in the Release Wizard's Update panel. This error can be caused by any of the following conditions:

- The file does not exist.
- The file is corrupted.
- The file is an incompatible media, for example, a media built by InstallShield Professional 5.x.

Error -7044

Message

Product Version is not specified in the Product Properties view.

Troubleshooting Information

You must specify a non-null value in the Product Version setting in the General Information view.

Error -7045

Message

File could not be found - "*path*\Corecomp.ini"

Troubleshooting Information

Corecomp.ini could not be found in the indicated location. If you have renamed or moved **Corecomp.ini**, place a copy in the indicated location with the file name **Corecomp.ini**. If you cannot restore **Corecomp.ini** in this way, run a repair of InstallShield.

Error -7126

Message

`Unable to build the setup.`

Description

There is not enough free disk space to build the setup.

Troubleshooting Information

The Media Wizard requires space in the system's temporary folder.Make sure you have enough free space on this drive, as well as in the build location.

Error -7127

Message

```
The Disk Images directory is use by another program.
```

Troubleshooting Information

This error can occur for any of the following reasons:

- The Disk Images or Disk1 folder that you are trying to build to may be open in Windows Explorer or DOS. Open a different folder in the Windows Explorer or DOS window, or close the window, and then rerun the media builder.
- The Disk Images or Disk1 folder is read-only. Right-click the folder in Windows Explorer and click Properties. If the folder is marked as read-only, uncheck the read-only check box and then rerun the media builder.
- A folder named bldback exists under the folder where the media is being built to. If such a folder exists, delete it, and then rerun the media builder.

Error -7275

Message

```
Certificate invalid.
```

Troubleshooting Information

The digital certificate that is specified for your project is invalid. Specify a valid certificate and build again.

Error -7276

Message

```
Certificate expired.
```

Troubleshooting Information

Your digital certificate has expired. Specify a current certificate and build again.

Error -7380

Message

```
Build canceled by the user.
```

Description

The build has been canceled by the user.

Troubleshooting Information

The user canceled the build process by clicking the Cancel button.

Error -9008

Message

```
Failed saving object property.
```

Troubleshooting Information

This error can occur if an object in the project is corrupted (for example, is missing Isrt.dll).

MSI/MSM Conversion Errors

The table below provides troubleshooting tips for each error that could be encountered while converting an installation package (.msi file) or merge module (.msm file) in the Open MSI/MSM Wizard.

The wizard also returns errors from accessing Windows Installer databases. For more information on those errors, see Build Errors and Warnings. You might receive undocumented, internal errors if the source .msi or .msm package is invalid.

Tip • *Before launching the Open MSI/MSM Wizard, validate the package with Orca or Msival2.exe, both part of the Microsoft Windows Installer Platform SDK.*

Table 12-4 • MSI and MSM Conversion Errors

Error Number	Message	Troubleshooting Tips
-6000	Unable to open file	The path to the .msi or .msm file might be invalid. The file might be open or in use by another program. The file might have been corrupted or might not be a valid Windows Installer database.

Table 12-4 • MSI and MSM Conversion Errors (cont.)

Error Number	Message	Troubleshooting Tips
-6001	Unsupported database schema	The Open MSI/MSM Wizard supports databases compatible with schema 30 or later only. To determine the schema of the .msi or .msm file, open it in Orca and on the **View** menu, click **Summary Information**. In some cases, you can work around this error simply by changing the **Schema** field in the **Edit Summary Information** dialog box.
-7000	Could not create record	If the record already exists, the wizard will not create a duplicate entry found in the file.
-7001 **-7002** **-7004**	Foreign key invalid	The wizard could not create a record because the record referenced by a foreign key in the .msi or .msm database was invalid or missing.
-7005	Unable to set string property	The Open MSI/MSM Wizard could not set a string value, probably because the field was a duplicate in the database file. This error might have resulted from an earlier failure to create a record (error -7000, -7001, -7002, or -7004).
-7006	Unable to set integer property	The Open MSI/MSM Wizard could not set an integer value, probably because the field was a duplicate in the database. This error might have resulted from an earlier failure to create a record (error -7000, -7001, -7002, or -7004).
-7008	Unable to set property	The Open MSI/MSM Wizard could not set a value, probably because the field was a duplicate in the database. This error might have resulted from an earlier failure to create a record (error -7000, -7001, -7002, or -7004).
-7009	Unable to create binary file	The wizard could not extract a binary file, an icon, or an .rtf file from the file and copy it to the new project location. Delete any existing file in the extraction directory. Ensure there is enough disk space and that you have the appropriate write privileges for the extraction directory.

Table 12-4 • MSI and MSM Conversion Errors (cont.)

Error Number	Message	Troubleshooting Tips
-7010	Output path for cab extraction does not exist	The wizard could not extract a .cab file using the path specified. Instead of typing the output path for the .cab file, click **Browse** to select the path and then launch the wizard again.
-7011	Cannot delete temporary .cab file	Unable to delete the temporary .cab file after extracting it from the installation package or merge module.
1001	Invalid source MSI database path	Ensure that the path to the .msi database is valid.
1017	End user aborted process	Allow the conversion process to complete.
100001	Unexpected error	The error message lists the function and table name associated with the error.
100002	Unable to extract file from the cabinet	Verify that the source is a valid .msi or .msm database.
200001	No output path was specified	This warning appears if the converter needs to extract binary files from the database tables or cabinet files, but an invalid output path was specified in the Data File Location box on the **File Locations** panel of the Open MSI/MSI Wizard.

Internal Errors

The following are internal errors. They should not be encountered when importing a valid .msi or .msm database. Before launching the Open MSI/MSM Wizard, validate the package with Orca or Msival2.exe, both part of the Microsoft Windows Installer Platform SDK.

The error message will display the database table, field, and value for which the conversion failed.

Table 12-5 • Internal Errors

Error Number	Message	Troubleshooting Tips
1002	Invalid target database path	Verify that your default project location path is valid. To verify the path, on the **Tools** menu, click **Options**. The default project location path is specified on the File **Locations** tab.
1003	Failed to open the source MSI/MSM database	Ensure that the database is not locked or open in another application.

Table 12-5 • Internal Errors (cont.)

Error Number	Message	Troubleshooting Tips
1004	Failed to open the target database	Ensure that the project file is not locked or open in another application. Verify that your default project location path is valid. To verify the path, on the **Tools** menu, click **Options**. The default project location path is specified on the File **Locations** tab.
1006	Failed to obtain table list for the source or target MSI/ MSM database	Verify that the source is a valid .msi or .msm database.
1007	Failed to obtain a field	This is generally caused by failure of the MsiRecordGetXXX API. Verify that the source is a valid Windows Installer database.
1009	MsiCreateRecord failure	Ensure that the project file is not locked or open in another application.
1010	Failed to update a field	This is generally caused by failure of the MsiRecordSetXXX API. Ensure that the project file is not locked or open in another application.
1011	Failed to insert a record into a table	Ensure that the project file is not locked or open in another application.
1012	Failed to extract binary data from the source MSI/MSM database	Delete any existing file in the extraction directory. Ensure that there is enough disk space and that you have the appropriate write privileges in the output path specified in the **Data File Location** box on the **File Locations** panel of the Open MSI/MSI wizard.
1013	Failed to stream a file into the target database	Ensure that the project file is not locked or open in another application.
1014	MsiDatabaseCommit failed	Ensure that the project file is not locked or open in another application.
5001	Failed to resolve a field value	This is generally caused by an unexpected database schema. See troubleshooting notes for error -6001.
5002	Incompatible field type between source and target MSI database	Verify that the source is a valid .msi or .msm database.

Windows Installer Run-time Errors



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*
- *MSI Database*
- *MSM Database*
- *QuickPatch*
- *Transform*

The following table contains a listing of Windows Installer errors that may occur during run time of an installation. The errors are related to the built-in InstallShield custom actions that are added automatically to InstallShield projects to support different functionality. For a list of the custom actions, see InstallShield Custom Action Reference.



Tip • *You can find detailed information—including resolution information—on some of the run-time errors in the Knowledge Base.*

Table 12-6 • Windows Installer Run-time Errors

Error Number	Message	Troubleshooting Information
27500	This setup requires Internet Information Server for configuring IIS Virtual Roots. Please make sure that you have IIS installed.	This error may occur if your project contains IIS data that you configured in the Internet Information Services view. If you encounter this error, ensure that the target system has IIS installed.
27501	This setup requires Administrator privileges for configuring IIS Virtual Roots.	This error may occur if your project contains IIS data that you configured in the Internet Information Services view. This error occurs if you do not have administrator privileges for configuring IIS virtual directories, but you are running an installation that configures IIS virtual directories.
27502	Could not connect to [2] [3]. [4]	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view. This error occurs if the installation was unable to connect to the specified database server.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27503	Error while retrieving version information from the specified database server.	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view. This error occurs if you specified the minimum version requirement for a database server on the Requirements tab for a SQL connection in the SQL Scripts view, but the installation could not determine what version of the database server exists on the target system.
27504	The specified database server does not meet the version requirements.	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view. This error occurs if the target system does not meet the minimum version requirements for a database server that is selected on the Requirements tab for a SQL connection in the SQL Scripts view. Note that in this scenario, this error occurs only if the **Allow installation to continue when minimum requirements are not met** check box on this tab is cleared.
27505	Fail to open a SQL script file.	This error may occur if your project contains a SQL script that you configured in the SQL Scripts view, and the run time fails to extract the SQL script file from the .msi package.
27506	Error while executing a SQL script.	This error may occur if your project contains a SQL script that you configured in the SQL Scripts view. This error occurs if the target database server encounters a SQL scripting error. To resolve this error, check the SQL script in your project and make any necessary changes.
27507	Browsing for SQL Servers requires that ODBC32.dll be installed and registered.	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view. To prevent this error from occurring, add the InstallShield prerequisite for MDAC to your installation project.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27508	Error installing COM+ application [2]. [3]	This error may occur if your project contains a COM+ application that you configured in the Component Services view. This error occurs if there are missing dependencies for COM+ component DLLs or incorrect settings for your COM+ application. To resolve this error, add the dependencies to your project or check the COM+ application in your project and make any necessary changes.
27509	Error uninstalling COM+ application [2]. [3]	This error may occur if your project contains a COM+ application that you configured in the Component Services view. This error occurs if the run time fails to delete the COM+ application from the system.
27510	Error installing COM+ application [2]. Could not load Microsoft(R) .NET class libraries. Registering .NET serviced components requires that Microsoft(R) .NET Framework be installed.	This error may occur if your project contains a COM+ application that you configured in the Component Services view. To prevent this error from occurring, consider adding the .NET Framework to your project. To learn how, see Adding .NET Framework Redistributables to Projects. As an alternative, you can use the Installation Requirements page in the Project Assistant to add a .NET Framework requirement to your project. If a target system does not have the appropriate version of the .NET Framework, the installation displays an error message, and the product is not installed.
27511	Could not execute SQL script file [2]. Connection not open: [3]	This error may occur if your project contains a SQL script that you configured in the SQL Scripts view. This error occurs if the connection is disconnected before the installation runs the SQL script.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27514	This installation requires a Microsoft SQL Server. The specified server [3] is a Microsoft SQL Server Desktop Engine or SQL Server Express.	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view. This error occurs if the target system has Microsoft SQL Server Desktop Engine or Microsoft SQL Server Express, and if the following check boxes are cleared on the Requirements tab for a SQL connection in the SQL Scripts view: • Allow installation to Microsoft SQL Server Desktop Engine/SQL Server Express • Allow installation to continue when minimum requirements are not met
27515	Error retrieving schema version from [2] [3]. Database: [4]. [5]	This error may occur if your project contains a SQL script that you configured in the SQL Scripts view, and if a version is specified in the Schema Version setting on the General tab for that SQL script. This error occurs if the target database server encounters a SQL script error when the run time attempts to retrieve the current schema version from the **InstallShield** table.
27516	Error writing schema version to [2] [3]. Database: [4]. [5]	This error may occur if your project contains a SQL script that you configured in the SQL Scripts view, and if a version is specified in the Schema Version setting on the General tab for that SQL script. This error occurs if the target database server encounters a SQL script error when the run time attempts to write a schema version to the **InstallShield** table.
27517	This installation requires Administrator privileges for installing COM+ applications. Log on as an administrator and then retry this installation.	This error may occur if your project contains a COM+ application that you configured in the Component Services view. This error occurs if you do not have administrator privileges for installing COM+ applications, but you are running an installation that includes COM+ applications.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27519	Error updating XML file [2]. [3]	This error may occur if your project contains XML references that you configured in the XML File Changes view. This error occurs if MSXML fails to save the target XML file. The [3] parameter refers to the error code that MSXML returned. Possible error codes are: • **XML_BAD_ENCODING**—The XML file contains a character that is not part of the specified encoding, which is defined on the Advanced tab for an XML file in the XML File Changes view. • **E_INVALIDARG**—The XML file does not have a valid file name. • **E_ACCESSDENIED**—A file system error occurred. • **E_OUTOFMEMORY**—The XML file could not be saved. Other values that can occur indicate that a file system error occurred.
27520	Error opening XML file [2]. [3]	This error may occur if your project contains XML references that you configured in the XML File Changes view. This error occurs if MSXML fails to load the target XML file. The [3] parameter refers to the error code that MSXML returned. A possible error codes is: • **S_FALSE**—The XML file load failed.
27521	This setup requires MSXML 3.0 or higher for configuring XML files. Please make sure that you have version 3.0 or higher.	This error may occur if your project contains XML references that you configured in the XML File Changes view. To prevent this error from occurring, consider using the Redistributables view to add an InstallShield prerequisite or merge module for MSXML to your project.
27524	Error loading NetApi32.DLL. The ISNetApi.dll needs to have NetApi32.DLL properly loaded and requires an NT based operating system.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if **NetApi32.dll** cannot be loaded on the target system.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27525	Server not found. Verify that the specified server exists. The server name can not be empty.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you do not enter an existing domain or server name on the LogonInfoNameAndServer dialog or the LogonInformation dialog.
27526	Unspecified error from ISNetApi.dll.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if an unexpected error is returned from a function in **NetApi32.dll**.
27527	The buffer is too small.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if the run time cannot allocate the amount of memory that is required for retrieving a list of all servers that are visible in a domain.
27528	Access denied. Check administrative rights.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if the user does not have access to the Windows user account information that is being configured.
27529	Invalid computer.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if the computer name that was entered is invalid.
27531	Unhandled exception.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if one of the function calls in **NetApi32.dll** throws an exception.
27532	Invalid user name for this server or domain.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you specify a user name that does not exist on a domain in the LogonInformation dialog or the LogonInfoNameAndServer dialog.
27533	The case-sensitive passwords do not match.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you enter different passwords in the Password setting and the Confirm Password setting on the LogonInfoCreateUser dialog.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27534	The list is empty.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if the LogonInfoListServers dialog needs to be displayed but no servers are found.
27535	Access violation.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if a memory access violation is encountered when information about user accounts or servers is retrieved and being processed.
27536	Error getting group.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if the installation encounters an error when retrieving information about a group account.
27537	Error adding user to group. Verify that the group exists for this domain or server.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you try adding membership of a user account to a group but the account type is invalid, the member is already a member of the group, or the member does not exist.
27538	Error creating user.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if the installation encounters a problem when creating the user account or adding the user to a group.
27539	ERROR_NETAPI_ERROR_NOT_PRIMARY returned from NetAPI.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you are trying to add a user account, give an existing user account membership, or delete a user account, but the operation is allowed only on the primary domain controller of the domain.
27540	The specified user already exists.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you are trying to create a new user account but the user account already exists.
27541	The specified group already exists.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you are trying to create a new user account but the group already exists.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27542	Invalid password. Verify that the password is in accordance with your network password policy.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you are creating a new user account but the password that you specified does not meet the password requirements that were established for the network. To resolve this error, enter an appropriate password.
27543	Invalid name.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you are giving an existing user account membership in a group, or if you are deleting a user account, but the user name that you specified could not be found.
27544	Invalid group.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you are giving a user account membership in a group, but the group that you specified does not exist.
27545	The user name can not be empty and must be in the format DOMAIN\Username.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if you are configuring a user account but you did not enter valid information for the User Name setting.
27546	Error loading or creating INI file in the user TEMP directory.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if there is a problem opening or creating a configuration .ini file that the InstallShield custom actions use for the creation or configuration of a Windows user account.
27548	Error deleting INI file containing new user information from the user's TEMP directory.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if there is a problem deleting the configuration .ini file that the InstallShield custom actions use for the creation or configuration of a Windows user account.
27549	Error getting the primary domain controller (PDC).	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if the domain controller for the specified domain could not be found.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27550	Every field must have a value in order to create a user.	This error may occur if you added to your project the ability to create or set a Windows user account. The error occurs if one of the settings on the LogonInfoCreateUser dialog is left blank.
27551	ODBC driver for [2] not found. This is required to connect to [2] database servers.	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view. This error occurs if the ODBC driver that is associated with the SQL connection does not exist on the target system. To prevent this error from occurring, consider creating an InstallShield prerequisite that installs the driver, and add the prerequisite to your project.
27552	Error creating database [4]. Server: [2] [3]. [5]	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view, and if the Create Catalog If Absent check box on the General tab for the connection is selected. This error occurs if the target database server encounters a SQL scripting error when creating the database catalog that is specified in the Catalog Name setting on the General tab for a SQL connection in the SQL Scripts view.
27553	Error connecting to database [4]. Server: [2] [3]. [5]	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view. This error occurs if the target database server encounters a SQL scripting error when connecting to the database catalog that is specified in the Catalog Name setting on the General tab for a SQL connection in the SQL Scripts view.
27554	Error attempting to open connection [2]. No valid database metadata associated with this connection.	This error may occur if your project contains a SQL connection that you configured in the SQL Scripts view. This error occurs if the connection is associated with no valid database metadata. To prevent this error from occurring, check the **ISSQLConnectionDBServer** table in the Direct Editor view, and make any necessary changes.

Table 12-6 • Windows Installer Run-time Errors (cont.)

Error Number	Message	Troubleshooting Information
27555	Error attempting to apply permissions to object [2]. System error: [3] ([4])	This error may occur if you used the custom InstallShield handling for securing files, folders, or registry keys in your project. To learn more about this functionality, see Securing Files, Folders, Registry Keys, and Windows Services in a Locked-Down Environment.

Setup.exe Return Values and Run-Time Errors (InstallScript Projects)



Project • *This information applies to InstallScript projects.*

Setup.exe may produce error messages if it cannot start properly. In most cases, you will encounter these messages when a severe error occurs. Rarely will your end users see these messages.

You can capture these return values when you call the **CreateProcess** API to launch **Setup.exe**, or use a batch file to launch **Setup.exe**.

When you get an error message, it appears in a message box. Every error message has a number. These are InstallScript system error messages, and there is no way to suppress them in your script. The following table describes each of these error messages:

Table 12-7 • Run-time Errors and Return Values for InstallScript Projects

Error/Return Value	Description
3010	The installation required a restart but the installation is not initiating the restart. In this case, the installation completed with BATCH_INSTALL set to a non-zero value, but the **System** function was not called. Note that **Setup.exe** returns this value only if the installation was successful and was not cancelled (that is, the installation did not end via the abort keyword). Otherwise, the appropriate error value or ISERR_SETUP_CANCELED is returned. The constant for this return value is ERROR_SUCCESS_REBOOT_REQUIRED.
1641	This return value occurs if the installation itself is restarting the machine because the **System** function was called—typically as a result of the **SdFinishReboot** function calling **System** internally. Note that **Setup.exe** returns this value only if the installation was successful and was not cancelled (that is, the installation did not end via the abort keyword). Otherwise, the appropriate error value or ISERR_SETUP_CANCELED is returned. The constant for this return value is ERROR_SUCCESS_REBOOT_INITIATED.
0	The installation exited with the exit keyword.

Table 12-7 • Run-time Errors and Return Values for InstallScript Projects (cont.)

Error/Return Value	Description
0x80042000	The installation exited with the abort keyword because the end user canceled the installation. The constant for this return value is ISERR_SETUP_CANCELED.
0x80040708	Unable to create required engine components. Check whether you have appropriate privileges to create COM components.  ***Note*** • *This error message may include an additional error number and error string indicating more specific error information returned by the operating system.*
-5001	Generic error
-5002	Failed reading media header.
-5003	Failed installing kernel.
-5004	Failed starting kernel.
-5005	Failed opening CAB.
-5006	Failed installing support.
-5007	Failed setting text substitution.
-5008	Failed initializing installation information.
-5009	Failed getting installation driver.
-5010	Failed initializing properties.
-5011	Failed running installation driver.
-5012	Failed uninstalling support.
-5013	Failed to extract file from setup boot file.
-5014	Failed to download file (occurs only when saving installation files during an Internet installation).
-5017	Could not clone the installation.
-6001	Failed starting the setup launcher.
-6002	Failed finding the setup launcher.

Table 12-7 • Run-time Errors and Return Values for InstallScript Projects (cont.)

Error/Return Value	Description
-6003	Failed loading the setup launcher.
-6004	Failed verifying the signature of setup launcher.
-6005	Failed installing the setup launcher to proper location.
-6006	Failed extracting setup launcher.

Typically, any time one of the error messages in the previous table is displayed, it is accompanied by one of the HRESULT values in the following table. These HRESULT values provide additional information about the cause of the error.

Table 12-8 • HRESULT Values Associated with Setup.exe Run-time Errors

HRESULT	Meaning
0x80041F40	Cannot find corecomp.in.i
0x80041F41	corecomp.ini is empty or corrupted.
0x80041F42	Cannot load the .dll file inIsCoGetClassObject.
0x80042328	Media is not signed.
0x80042329	Certificate is invalid.
0x8004232A	Digital signature is invalid.
0x8004232B	Space is insufficient.
0x80042A94	The specified opType's main opsequence is not present in the log, such as when an old file is opened with the new engine, and the new (engine) code tries to get an opsequence for the newly introduced opType.
0x80042A95	Failed to (re)construct a feature log's full ID.
0x80042A96	A log operation was attempted without opening the log file.
0x80042A97	Internal opsequence structure initialization failed.
0x80042A98	General failure in Opsequence::GetNext
0x80042A99	General failure in Opsequence::Pop
0x80042A9A	General failure in Opsequence::insert_sequencetuple
0x80042A9B	General failure in Opsequence::delete_sequencetuple

Table 12-8 • HRESULT Values Associated with Setup.exe Run-time Errors (cont.)

HRESULT	Meaning
0x80042A9C	Property initialization failure - could occur when setting/getting a property in the LogDB or Feature object.

Setup.exe Return Values and Run-Time Errors (Basic MSI and InstallScript MSI Projects)



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Note that project-specific differences are explained where applicable.

The table below lists the errors that might occur when **Setup.exe** from a Basic MSI or InstallScript MSI project is run. For **Setup.exe** errors that might occur for an InstallScript project, see Setup.exe Return Values and Run-Time Errors (InstallScript Projects).

You can capture these return values when you call the **CreateProcess** API to launch **Setup.exe**, or use a batch file to launch **Setup.exe**.



Note • *If an error occurs on a non-English system, the error strings may not be displayed in English.*

Table 12-9 • Setup.exe Run-time Errors and Return Values

Error or Status Number	Message	Troubleshooting Tips
-4	Invalid command line.	 ***Project*** • *This is applicable to InstallScript MSI projects.* Verify that a valid command line is passed to **Setup.exe**.
-3	The installation exited because the end user canceled the installation.	 ***Project*** • *This is applicable to InstallScript MSI projects.*
-1	General error.	 ***Project*** • *This is applicable to InstallScript MSI projects.*
0	Program terminated successfully.	Not applicable.

Table 12-9 • Setup.exe Run-time Errors and Return Values (cont.)

Error or Status Number	Message	Troubleshooting Tips
401	String variable is not large enough for string. InstallShield was attempting to copy a text string into a string variable. The text string was larger than the length declared for that string variable.	Check the declared length of the string variable. Increase the length to the maximum allowed value.
1150	Setup has detected an incompatible version of Windows. Click OK and relaunch the setup on Windows 95, Windows NT 4.0, or later.	Windows Installer is compatible with NT 4.0 and later, and Windows 9x and later. Check your version of Windows and upgrade if necessary.
1151	Error writing to the temporary location	To write to the temporary location, the environment variable TEMP must be set. Verify that the Temp folder exists and has enough disk space to accommodate the setup. If there are files in the Temp folder, delete them and rerun Setup.exe.
1152	Error extracting <file name> to the temporary location	Check to see that you are able to write to the Temp folder (see errors above). If the Temp folder is valid, there may be corrupted files in the setup. Check the files to ensure none are corrupted and rerun Setup.exe.
1153	Error reading setup initialization file	The Setup.ini file must be located in the same folder as Setup.exe. If not, move Setup.ini to that location.
1154	Installer not found in <path>	Windows Installer may not have been properly installed, or you may have an older version. Reinstall if necessary.
1155	File <file name> not found	Make sure the .msi file exists. If so, make sure it is located in the same folder as **Setup.exe**. You may not see the .msi file if you chose to compress it into **Setup.exe**.
1156	Internal error in Windows Installer	Windows Installer was unsuccessfully installed. Run the **InstMsiW.exe** file (for Windows NT and 2000) or **InstMsiA.exe** (for Windows 9x) to reinstall.
1157	Failed to launch Msiexec.exe.	Make sure you distribute the correct version of Windows Installer for the target platform. Check the syntax on your Msiexec.exe command-line arguments.

Table 12-9 • Setup.exe Run-time Errors and Return Values (cont.)

Error or Status Number	Message	Troubleshooting Tips
1158	Error populating strings.	Verify that all strings in Setup.ini and any language-specific INI files in the Disk1 folder (such as 0x0409.ini) are valid.
1201	Setup needs <amount> KB free space in <folder>. Please free up some space and try again.	There is insufficient disk space in your target location. Please make sure there is at least 10 MB of free space in the drive where the setup is set to install.
1202	You do not have sufficient privileges to complete this installation for all users of the machine. Log on as an administrator and then retry this installation.	In Windows NT 4.0 and 2000, you must have administrative rights to complete this installation.
1203	Invalid command-line parameters.	Double-check the command-line statement used to launch Setup.exe.
1208	ANSI code page for <language> is not installed on the system and therefore setup cannot run in the selected language. Run the setup and select another language.	Run the setup and select another language.
1603	General Windows Installer engine error. Increase DiskSpace requirement in Setup.ini and try again.	 ***Project*** • *This is applicable to Basic MSI projects.* Reinstall Windows Installer.
1611	Unable to extract the file <filename>.	This error occurs when a file compressed inside **Setup.exe** cannot be extracted. Verify that there is sufficient disk space available in the Temp folder (or, if not, in the Windows or WinNT folder), and that Setup can write to those folders.
1614	An error occurred while downloading the file <filename>.	If the URL is incorrect, click **Cancel** and enter the correct URL. If the URL is correct, verify that an active Internet connection is available.

Table 12-9 • Setup.exe Run-time Errors and Return Values (cont.)

Error or Status Number	Message	Troubleshooting Tips
1621	Failed to verify signature of file <filename>.	The file was downloaded, but the signature could not be verified. The file might be corrupted, it might have been signed by a different company than originally signed it, or it might not be signed. Verify that the file Wintrust.dll exists on the target system.
1627	Unable to save file: <filename>.	Ensure that the specified file does not already exist, and that the target system has sufficient hard drive space.
1628	Failed to complete script based install.	 ***Project*** • *This is applicable to InstallScript MSI projects.*
1629	Invalid command line.	Double-check the command-line statement used to launch Setup.exe.
1641	The installation was successful and it required a restart.	 ***Project*** • *This is applicable to InstallScript MSI projects.* If the Windows Installer engine returns ERROR_SUCCESS (0), the **Setup.exe** file may return ERROR_SUCCESS_REBOOT_INITIATED (1641). This return value occurs if the installation itself is restarting the machine because the **System** function was called—typically as a result of the **SdFinishReboot** function calling **System** internally.
1670	Unable to load module [1], Error Code: [2]	[1] is the full path to the .dll file that the installation is attempting to load (**ISSetup.dll** in all cases). [2] is the Windows error returned from **LoadLibrary()**. This error occurs if **ISSetup.dll** is not in the Disk1 folder. It also may occur if **ISSetup.dll** is compressed in **Setup.exe** if a compressed media is being built. To resolve this error, try rebuilding the release.
1700	An error occurred initializing the InstallScript engine.	 ***Project*** • *This is applicable to InstallScript MSI projects.* This error indicates that a problem occurred when the InstallScript engine was being loaded.

Table 12-9 • Setup.exe Run-time Errors and Return Values (cont.)

Error or Status Number	Message	Troubleshooting Tips
1701	Unable to extract InstallScript engine support files to temp location.	***Project*** • *This is applicable to InstallScript MSI projects.* This error occurs if one or more files could not be extracted from the **ISSetup.dll** file to a temporary directory. To troubleshoot this error, create a Windows Installer log file to obtain more information. The log file has the corresponding Windows error number and other diagnostic information that may help troubleshoot this issue.
1919	Error configuring ODBC data source. Verify that the file exists and that you can access it.	Machines running Windows 95 do not have the ODBC core. You need to install MDAC before installing any package with an ODBC driver.
3010	The installation was successful and a restart is required to complete the installation.	***Project*** • *This is applicable to InstallScript MSI projects.* If the Windows Installer engine returns ERROR_SUCCESS (0), the **Setup.exe** file may return ERROR_SUCCESS_REBOOT_REQUIRED (3010). This indicates that installation required a restart but the installation is not initiating the restart. In this case, the installation completed with BATCH_INSTALL set to a non-zero value, but the **System** function was not called.

Setup.exe Return Values and Run-Time Errors (Advanced UI and Suite/Advanced UI Projects)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The table below lists the errors that might occur when an end user runs an Advanced UI or Suite/Advanced UI **Setup.exe** installation. The Type of Code column in this table indicates the following information:

- **Exit code only**—The installation status is returned by the Advanced UI or Suite/Advanced UI **Setup.exe** file.
- **Error code/exit code**—The installation status is processed for evaluated error conditions, and the status is returned by the Advanced UI or Suite/Advanced UI **Setup.exe** file.
- **Logged only**—The installation status is logged when debug logging is enabled. (Debug logging is enabled when the `/debuglog` command-line parameter is passed to the Advanced UI or Suite/Advanced UI **Setup.exe** file to launch the Advanced UI or Suite/Advanced UI installation. For more information on command-line parameters, see Advanced UI and Suite/Advanced UI Setup.exe Command-Line Parameters.)



***Note** • Some of the error messages may be displayed before any string entries are available to the running Advanced UI or Suite/Advanced UI* **Setup.exe** *file. Therefore, in such cases (for example, Setup.xml is missing, another instance of the Advanced UI or Suite/Advanced UI installation is already running, or condition validation failed), those particular resulting error messages are displayed in English. Other errors that occur before language selection are typically displayed in the default setup language.*

Table 12-10 • Setup.exe Run-time Errors and Return Values

Status Code	Type of Code	Message	Troubleshooting Tips
0x80040701	**Exit code only**	Failed to read or initialize setup state information; missing SuiteId attribute on setup.xml setup element. Contact your vendor for assistance.	The value of the SuiteID attribute in the **Setup.xml** file indicates the Suite GUID, as defined in the General Information view of the project. If this GUID is somehow removed from the **Setup.xml** file, this error may occur at run time. To resolve this error, try rebuilding the Advanced UI or Suite/Advanced UI installation.
0x80040702	**Exit code only**	No UI DLL was loaded and initialized, possibly due to a missing UI resource. Contact your vendor for assistance.	This exit code occurs if the UIResource definition in the project or the **Setup.xml** file is missing or invalid (for example, for a corrupted project).
0x80040705	**Exit/error code**	MD5 check failed for the current file to be staged. The file may be corrupted.	An Advanced UI or Suite/Advanced UI installation uses MD5 checks the integrity of the packages in the Advanced UI or Suite/Advanced UI installation. If the MD5 check fails, this error occurs. To resolve this error, try rebuilding the Advanced UI or Suite/Advanced UI installation.

Table 12-10 • Setup.exe Run-time Errors and Return Values (cont.)

Status Code	Type of Code	Message	Troubleshooting Tips
0x80040706	**Exit code only**	No setup.xml file was found in the running suite setup. Contact your vendor for assistance.	This status code occurs if the **Setup.xml** resource could not be found.
0x80040707	**Logged only**	Failed to extract a file that is necessary to run this setup. Contact your vendor for assistance.	The log includes this status code if a resource that is listed in the Resources section of the **Setup.xml** file could not be extracted from the running Advanced UI or Suite/Advanced UI **Setup.exe** file. Check for build errors and the Support Files view for missing resources, and add them as needed.
0x80040708	**Logged only**	Failed to copy or download a file that is necessary to run this setup. Contact your vendor for assistance.	The log includes this status code if a resource that is listed in the Resources section of the **Setup.xml** file could not be copied or downloaded. Check for build errors and the Support Files view for missing resources, and add them as needed.
0x80040709	**Error code only**	Failed to extract a file while staging. Either the file is not present in the stream or the file could not be written to the target machine.	This error occurs if the Advanced UI or Suite/Advanced UI installation fails to stage a file that is streamed into the Advanced UI or Suite/Advanced UI **Setup.exe** file. Either the installation is corrupted or the file was missing at build time.

Table 12-10 • Setup.exe Run-time Errors and Return Values (cont.)

Status Code	Type of Code	Message	Troubleshooting Tips
0x8004070A	**Exit/error code**	The setup password is incorrect or was not entered. The setup cannot proceed.	The Setup.exe tab for a release in the Releases view of an Advanced UI or Suite/Advanced UI project lets you specify whether you want to password protect the setup launcher; if you do, you can also specify the password that end users need to enter in order to run the installation. This error occurs if there is no prompt for a password from the user interface during the preinstall phases of the installation. This can occur if a custom UI DLL is used and the end user tries to avoid the password prompt. This can also occur if an end user launches a password-protected Advanced UI or Suite/Advanced UI installation silently from the command line but does not include a password on the command line.
0x8004070B	**Exit/error code**	The setup command line is invalid. The setup cannot proceed.	This error occurs if you pass an invalid command-line parameter to the Advanced UI or Suite/Advanced UI **Setup.exe** file while launching it from the command line. For a list of valid command-line parameters, see Advanced UI and Suite/Advanced UI Setup.exe Command-Line Parameters.
0x8004070C	**Exit/error code**	The setup could not start in a temporary folder. Contact your vendor for assistance.	This error occurs if the Temp folder on the target system is full. To resolve this error, consider freeing space in the Temp folder.

Table 12-10 • Setup.exe Run-time Errors and Return Values (cont.)

Status Code	Type of Code	Message	Troubleshooting Tips
0x8004070D	**Exit/error code**	The destination folder that was entered for a staging-only operation could not be created.	When you are running an Advanced UI or Suite/Advanced UI installation with the `/stage_only` command-line parameter, the Advanced UI or Suite/Advanced UI installation attempts to copy its packages to a directory that is specified by the user. The directory must exist; if it does not, the installation encounters this error. To resolve this error, ensure that the path that you specify on the BrowseStageFolder wizard page exists. If it does not, create it and then rerun the installation.
0x8004070E	**Exit/error code**	An unknown failure occurred attempting to cache a file from either a remote server or a local path. Contact your vendor for assistance.	If the file is located on a Web server, ensure that it is available on the machine where the installation is encountering this error. If the file is located on source media, ensure that the source media is available and valid. Check a debug log for any additional information that might be available.
0x8004070f	**Exit/error code**	An invalid condition was encountered while parsing **setup.xml**. Create a debug log to see if further information is available. The condition needs to be corrected in the IDE for the suite to launch successfully.	This error occurs if an invalid condition was configured. To troubleshoot this error, check the debug log file for additional information. To learn more, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation. To resolve this error, edit the invalid condition in your Advanced UI or Suite/Advanced UI project.

Table 12-10 • Setup.exe Run-time Errors and Return Values (cont.)

Status Code	Type of Code	Message	Troubleshooting Tips
0x80040710	**Error Code**	The setup was aborted because the suite requirements defined in abort conditions were not met.	This error occurs if the exit conditions that were defined in the General Information view were not met at run time on the target system. If this is unexpected, you may need to modify your exit conditions. To learn more, see Defining Exit Conditions for an Advanced UI or Suite/Advanced UI Installation
0x80040711	**Exit/error code**	Another instance of the suite installation is already running. Please wait for the other instance to finish and then try again.	Only one instance of an Advanced UI or Suite/Advanced UI installation can run on a target system at a time.
0x80040712	**Exit/error code**	Failed to load ISSetup.dll to launch an InstallScript parcel.	This error may occur if one or more of the files for an InstallScript package that is included in the project are not available to InstallShield at build time. To resolve this error, ensure that all of the files in your InstallScript package are available at build time.
0x80040713	**Exit/error code**	Failed to call ISSetup.dll to launch an InstallScript parcel. Ensure the InstallScript setup contained in the parcel was built with a version of InstallShield that supports InstallScript parcels.	This error may occur if you include in your project an InstallScript package that was built in InstallShield 2012 Spring or later. For additional requirements, see Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project.
0x80040714	**Exit/error code**	Failed to launch EXE. This error can occur when attempting to launch EXE actions, elevated EXE parcels, or Windows Features support on Windows Vista, 2008, 7, or 2008 R2 and the EXE fails to launch. Check a debug log as this may contain more detailed error information.	To troubleshoot this error, check the debug log file for additional information. To learn more, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.

Table 12-10 • Setup.exe Run-time Errors and Return Values (cont.)

Status Code	Type of Code	Message	Troubleshooting Tips
0x80040715	**Exit/error code**	Failed to load DLL. This error can occur when attempting to load a DLL for DLL actions or Windows Features support on Windows 8 or newer versions of Windows. Check a debug log as this may contain more detailed error information.	This error may occur if the file that a DLL action in your project needs to call is missing or could not be loaded. This may also occur if the installation is attempting to enable Windows roles or features on a target system. To troubleshoot this error, check the debug log file for additional information. To learn more, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.
0x80040716	**Exit/error code**	Failed to locate or load support files needed to run a PowerShell action. Check a debug log as this may contain more detailed error information.	This error may occur if the file that a PowerShell action in your project needs to call is missing or could not be loaded. To troubleshoot this error, check the debug log file for additional information. To learn more, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.
0x80040718	**Exit/error code**	Failed to call InstallScript function in InstallScript action. Check a debug log as this may contain more detailed error information.	This error occurs because the installation could not call an InstallScript function in an InstallScript action. To troubleshoot this error, check the debug log file for additional information. To learn more, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.
0x8004071A	**Error code**	The path specified for a suite debug log does not exist or is invalid. Specify a valid file path when passing /debuglog with a file path. The suite error returned in this case is 1622.	Ensure that you specify a valid path for the debug log file.

Table 12-10 • Setup.exe Run-time Errors and Return Values (cont.)

Status Code	Type of Code	Message	Troubleshooting Tips
0x8004071B	**Exit/error code**	A resource required to install a web deploy parcel was not found. Rebuild the setup and try again.	This error may occur if the file that a Web Deploy package in your project requires is missing. To troubleshoot this error, check the debug log file for additional information. To learn more, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.
0x8004071C	**Exit/error code**	A resource required to call a CLR action is missing or failed to load. Check a debug log as this may contain more detailed error information.	This error may occur if the file that a managed-code action in your project needs to call is missing or could not be loaded. To troubleshoot this error, check the debug log file for additional information. To learn more, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.
0x8007007f	**Error code only**	The specified procedure could not be found.	This error is system error 127. This error may occur if the Advanced UI or Suite/Advanced UI installation is run on a system that does not meet the minimum requirements for Windows Installer: ● To run an Advanced UI or Suite/Advanced UI installation on a target system, the target system must have Windows Installer 3.1 or later. To avoid this error on these systems, you may want to create an exit condition that prevents the entire Advanced UI or Suite/Advanced UI installation from running on systems that do not have Windows Installer 4.5 or later. To learn more, see Defining Exit Conditions for an Advanced UI or Suite/Advanced UI Installation.

Table 12-10 • Setup.exe Run-time Errors and Return Values (cont.)

Status Code	Type of Code	Message	Troubleshooting Tips
0x8007007f (cont.)			• If the Advanced UI or Suite/Advanced UI installation contains packages that run with transaction processing, the target system must have Windows Installer 4.5 or later. To avoid this error on these systems, you may want to create an eligibility condition for each package in the transaction to prevent the Advanced UI or Suite/Advanced UI installation from launching those packages on systems that have earlier versions of Windows Installer. For more information, see Building Conditional Statements in Advanced UI and Suite/Advanced UI Projects. This error may also occur if the Advanced UI or Suite/Advanced UI installation has trouble installing an InstallScript package because the package was built in InstallShield 2012 or earlier. For more information, see Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project. In addition, this error may occur if there is a problem with an Advanced UI or Suite/Advanced UI extension condition. If your project includes an extension condition, check to ensure that the DLL for the extension condition has the proper exported entry points.

Visual Studio Project Import Errors and Warnings

This table lists the errors and warnings that might occur when you do any of the following:

- Convert a Visual Studio setup project (.vdproj) to an InstallShield Basic MSI project (.ism).

- Convert a Visual Studio merge module project (.vdproj) to an InstallShield Merge Module project (.ism).
- Import a Visual Studio setup or merge module project (.vdproj) into an InstallShield Basic MSI or Merge Module project (.ism).

Table 12-11 • Visual Studio Project Import Errors and Warnings

Error or Warning Number	Message	Troubleshooting Tips
-9000	An unknown exception occurred.	Check the Knowledge Base for information about this error, or request technical support.
-9001	An unknown COM exception occurred.	Check the Knowledge Base for information about this error, or request technical support.
-9002	An error occurred while loading the project %1.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, or request technical support.
-9003	An error occurred while creating the project %1.	This error occurs if InstallShield cannot create a Basic MSI project from the Visual Studio setup project or a Merge Module project from the Visual Studio merge module project. InstallShield tries to create the project in the same folder that contains the Visual Studio project file (.vdproj). To resolve this error, see if the folder that contains the Visual Studio project file is read-only, and change it to writable.
-9004	An error occurred while saving the project %1.	This error occurs if InstallShield cannot create a Basic MSI project from the Visual Studio setup project or a Merge Module project from the Visual Studio merge module project. InstallShield tries to save the project in the same folder that contains the Visual Studio project file (.vdproj). To resolve this error, see if the folder that contains the Visual Studio project file is read-only, and change it to writable.
-9005	An error occurred while converting the project.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9006	An error occurred while determining the project type.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9007	An error occurred while parsing the section '%1'.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9008	No open brace found for the section %1 under %2.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9009	No close brace found for the section %1 under %2.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9010	An invalid entry %1 found in the section %1 under %2.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9011	An error occurred while adding the property %1 to the section %2.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9012	An error occurred while adding the section %1 to the section %2.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9013	The section %1 does not exist.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9014	An error occurred while converting the product properties.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9015	An error occurred while converting the files.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9016	The file type %1 is invalid. The file %2 was not converted.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9017	An error occurred while converting the file %1. Type: %2	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9018	An error occurred while converting the features.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9019	An error occurred while converting the feature %1.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9020	An error occurred while converting the folders.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9021	An error occurred while converting the folder %1. Type: %2	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9022	The folder type %1 is invalid. The folder %2 was not converted.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9023	An error occurred while converting the custom actions.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9024	An error occurred while converting the custom action %1. Type: %2	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9025	The custom action type %1 is invalid. The custom action %2 was not converted.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9026	A component for the source file %1 of the custom action %2 was not found.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9027	An error occurred while converting the file types.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9028	An error occurred while converting the file extension %1. Type: %2	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9029	The file extension type %1 is invalid. The file extension %2 was not converted.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9030	An error occurred while converting the verb %1. Type: %2	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.
-9031	The verb type %1 is invalid. The verb %2 was not converted.	This error may occur if the Visual Studio project file is corrupted or if InstallShield cannot correctly read the Visual Studio project file. To resolve this error, request technical support.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9032	A command is not specified for the file extension %1. The file extension was not converted.	This warning occurs if the Command property for the specified file type was left blank in the File Types Editor in Visual Studio. The warning alerts you that InstallShield could not configure the file extension during the conversion process. To resolve this issue, specify a command for the file type in Visual Studio, and then convert the Visual Studio setup project to an InstallShield project. Otherwise, you can add and configure the file extension in the File Types area of the Components view in InstallShield.
-9033	A component for the command %1 of the file extension %2 not found. The file extension was not converted.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9034	An error occurred while converting the registries.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9035	An error occurred while converting the registry key %1. Type: %2	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9036	The registry key type %1 is invalid. The registry key %2 was not converted.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9037	An error occurred while converting the registry value %1. Type: %2	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9038	The registry value type %1 is invalid. The registry value %2 was not converted.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9039	An error occurred while converting the shortcuts.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9040	An error occurred while converting the shortcut %1. Type: %2	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9041	The shortcut type %1 is invalid. The shortcut %2 was not converted.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9042	The shortcut target %1 is invalid. The shortcut %2 was not converted.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9043	An error occurred while converting the launch conditions.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9044	An error occurred while converting the launch condition %1. Type: %2	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9045	The launch condition type %1 is invalid. The launch condition %2 was not converted.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9046	An error occurred while converting the locators.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9047	An error occurred while converting the locator %1. Type: %2	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9048	The locator type %1 is invalid. The locator %2 was not converted.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9049	An error occurred while converting the product icon.	This error may occur if the Visual Studio setup or merge module project file is corrupted or if InstallShield cannot correctly read the Visual Studio setup project file. To resolve this error, request technical support.
-9050	The project language '%1' is not installed in InstallShield. The language was not converted.	This warning occurs if you convert a Visual Studio project that includes support for a language, but InstallShield does not have built-in support for that language.  ***Edition*** • *The Premier edition of InstallShield includes multilanguage support, but the Professional edition does not.*
-9052	File type '%1' specifies no extensions. The file type was not converted.	This warning occurs if the Extensions property for the specified file type was left blank in the File Types Editor in Visual Studio. The warning alerts you that InstallShield could not configure the file extension during the import or conversion process. To resolve this issue, specify one or more file extensions for the file type in Visual Studio, and then convert or import the Visual Studio project into an InstallShield project. Otherwise, you can add and configure the file extension in the Advanced Settings area of the Components view in InstallShield after you have converted or imported the Visual Studio project.
-9053	Launch condition '%1' specifies no conditions. The launch condition was not converted.	This warning occurs if the Condition property for the specified launch condition was left blank in the Launch Conditions Editor in Visual Studio. The warning alerts you that InstallShield could not configure the launch condition during the import or conversion process. To resolve this issue, specify a condition in Visual Studio, and then convert or import the Visual Studio project into an InstallShield project. Otherwise, you can add and configure the launch condition in the General Information view in InstallShield after you have converted or imported the Visual Studio project.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9054	File search '%1' specifies no file names. The file search was not converted.	This warning occurs if the FileName property for the specified file search was left blank in the Launch Conditions Editor in Visual Studio. The warning alerts you that InstallShield could not configure the file search during the import or conversion process. To resolve this issue, specify the file name for the file search in Visual Studio, and then convert or import the Visual Studio project into an InstallShield project. Otherwise, you can add and configure the file search in the System Search view in InstallShield after you have converted or imported the Visual Studio project.
-9055	Feature '%1' already exists in the InstallShield project. The feature was not converted.	This warning occurs if you have a Visual Studio project that contains a particular feature and you import that project into an InstallShield project that already contains that feature. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once.
-9056	Directory '%1' already exists in the InstallShield project. The folder '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a particular folder and you import that project into an InstallShield project that already contains that folder. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the Files and Folders view to ensure that the folder that is identified in the warning message is not missing from your InstallShield project.
-9057	File key '%1' already exists in the InstallShield project. The file '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a file that is associated with a particular file key and you import that project into an InstallShield project that already contains a file with that same file key. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the Files area of the Components view to ensure that the file that is identified in the warning message is not missing from your InstallShield project. Note that the Direct Editor view shows the file key that is associated with each file in your project. The file keys are the primary keys of the **File** table in the .msi database; the **File** table cannot contain duplicate file keys.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9058	Shortcut key '%1' already exists in the InstallShield project. The shortcut '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a shortcut that is associated with a particular shortcut key and you import that project into an InstallShield project that already contains a shortcut with that same shortcut key. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the Shortcuts view to ensure that the shortcut that is identified in the warning message is not missing from your InstallShield project.
-9059	Extension '%1' already exists in the InstallShield project. The file type '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a particular file extension and you import that project into an InstallShield project that already contains that same file extension. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the Advanced Settings area of the Components view to ensure that the file extension that is identified in the warning message is not missing from your InstallShield project.
-9060	Registry key '%1' already exists in the InstallShield project. The registry entry 'Key: %2 Value: %3 Data: %4' was not converted.	This warning occurs if you have a Visual Studio project that contains a particular registry entry and you import that project into an InstallShield project that already contains that same registry entry. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the Registry view to ensure that the registry entry that is identified in the warning message is not missing from your InstallShield project.
-9061	Custom action '%1' already exists in the InstallShield project. The custom action named '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a particular custom action and you import that project into an InstallShield project that already contains a custom action with that same name. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the Custom Actions and Sequences view to ensure that the custom action that is identified in the warning message is not missing from your InstallShield project.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9062	Launch condition '%1' already exists in the InstallShield project. The launch condition named '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a particular launch condition and you import that project into an InstallShield project that already contains a launch condition with that same name. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the Install Condition setting in the General Information view to ensure that the launch condition that is identified in the warning message is not missing from your InstallShield project.
-9063	RegLocator '%1' already exists in the InstallShield project. The registry search '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a particular registry search and you import that project into an InstallShield project that already contains a registry search with that same name. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the System Search view to ensure that the registry search that is identified in the warning message is not missing from your InstallShield project.
-9064	DrLocator '%1' already exists in the InstallShield project. The file search '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a particular file or folder search and you import that project into an InstallShield project that already contains a file or folder search with that same name. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the System Search view to ensure that the file or folder search that is identified in the warning message is not missing from your InstallShield project.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9065	CompLocator '%1' already exists in the InstallShield project. The Windows Installer search '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a particular file or folder search and you import that project into an InstallShield project that already contains a file or folder search with that same name. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the System Search view to ensure that the file or folder search that is identified in the warning message is not missing from your InstallShield project.
-9070	An error occurred while converting the prerequisites.	This error occurs if you import a Visual Studio setup project that contains a prerequisite but InstallShield encounters a problem when trying to map the Visual Studio prerequisite to an equivalent InstallShield prerequisite. To resolve this error, request technical support, or consider creating an InstallShield prerequisite that installs the required redistributable, and add that InstallShield prerequisite to your project. For more information, see Defining InstallShield Prerequisites and Working with InstallShield Prerequisites that Are Included in Installation Projects.
-9071	InstallShield has no equivalent InstallShield prerequisite for '%1'. The prerequisite was not converted.	This warning occurs if you import a Visual Studio setup project that contains a prerequisite but InstallShield does not have an equivalent InstallShield prerequisite. To resolve this warning, consider creating an InstallShield prerequisite that installs the required redistributable, and add that InstallShield prerequisite to your project. For more information, see Defining InstallShield Prerequisites and Working with InstallShield Prerequisites that Are Included in Installation Projects.
-9074	An error occurred while converting the project outputs.	This error occurs if you convert a Visual Studio setup or merge module project that contains one or more project outputs but InstallShield encounters a problem when incorporating the project outputs into the InstallShield project. To resolve this error, request technical support.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9075	An error occurred while converting the project output %1. Type: %2	This error occurs if you convert a Visual Studio setup or merge module project that contains one or more project outputs but InstallShield encounters a problem when incorporating the project outputs into the InstallShield project. To resolve this error, request technical support.
-9076	The project output type %1 is invalid. The project output %2 was not converted.	This error occurs if you convert a Visual Studio setup or merge module project that contains one or more project outputs but one of the project outputs is invalid. To resolve this error, review the project output in your Visual Studio project, and try to resolve any issues with it. Then convert the Visual Studio project to an InstallShield project.
-9077	File key '%1' already exists in the InstallShield project. The project output '%2' was not converted.	This warning occurs if you have a Visual Studio project that contains a project output that has a file that is associated with a particular file key, and you import that project into an InstallShield project that already contains a file with that same file key. This scenario may occur if you import the same Visual Studio project into your InstallShield project more than once. If you encounter this warning, check the Files and Folders view to ensure that the file that is identified in the warning message is not missing from your InstallShield project. Note that the Direct Editor view shows the file key that is associated with each file in your project. The file keys are the primary keys of the **File** table in the .msi database; the **File** table cannot contain duplicate file keys.
-9078	The InstallShield project is not in a Visual Studio solution. The project output '%1' was not converted.	This error occurs if your InstallShield project is open in InstallShield, but not from within Visual Studio, and you try to import a Visual Studio project that contains project outputs into the InstallShield project. To resolve this error, open your InstallShield project from within Visual Studio, and then import the Visual Studio project into it.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9079	The Visual Studio project '%1' could not be found in the current Visual Studio solution. The project output '%2' was not converted.	If you want to import into an InstallShield project a Visual Studio setup or merge module project that contains a project output, the Visual Studio project must be in the same solution as all of its project dependencies. Otherwise, this error occurs when you try to import the Visual Studio project. To resolve this error, ensure that your InstallShield project is in the same Visual Studio solution that contains the Visual Studio setup or merge module project that you are importing. In addition, ensure that the solution also includes all of the other Visual Studio projects that are dependencies of the Visual Studio setup or merge module project. Then you can import the Visual Studio setup or merge module project into your InstallShield project.
-9080	The project output group '%1' could not be found in the Visual Studio project '%2'. The project output '%3' was not converted.	This error occurs if you convert a Visual Studio setup or merge module project that contains a project output group but InstallShield encounters a problem when incorporating a project output of that group into the InstallShield project. To resolve this error, request technical support.
-9081	File '%1' is specified to be excluded from a setup or merge module. The file was not converted.	This warning occurs if you import into an InstallShield project a Visual Studio setup or merge module project that contains a file whose Exclude property is set to True. If you want the specified file to be excluded from your InstallShield project, you can ignore this warning. If you want the specified file to be included in your InstallShield project, you can use the Files and Folders view in InstallShield to add it. For more information, see Adding to Your Project Files to Be Installed on Target Systems.
-9082	Project output '%1' is specified to be excluded from a setup or merge module. The project output was not converted.	This warning occurs if you import into an InstallShield project a Visual Studio setup or merge module project that contains a file in a project output, and True is selected for the Exclude property of that file. If you want the specified file to be excluded from your InstallShield project, you can ignore this warning. If you want the specified file to be included in your InstallShield project, you can use the Files and Folders view in InstallShield to add it. For more information, see Adding References to Visual Studio Solutions.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9083	Visual Studio default launch conditions are not supported in InstallShield. The launch condition named '%1' was not converted.	This warning occurs if you import into InstallShield project a Visual Studio setup or merge module project that contains a default launch condition. If you want the specified launch condition to be excluded from your InstallShield project, you can ignore this warning. If you want to add a launch condition to your InstallShield project to check the target system for the default launch condition that Visual Studio added to the original project, you can use the System Search view in InstallShield to add it. For more information, see Adding a System Search.
-9086	MinDate '%1' for the file search named '%2' is not valid. The MinDate property was not converted.	This warning occurs if you attempt to import or convert a Visual Studio setup that has a file search with an invalid value for the MinDate property. To resolve this warning: **1.** Open the **System Search** view in InstallShield. **2.** Right-click the file search and then click **Modify**. The **System Search Wizard** opens. **3.** On the second panel of this wizard, click the **Details** button. The **File Details** dialog box opens. **4.** Enter the appropriate details.
-9087	MaxDate '%1' for the file search named '%2' is not valid. The MaxDate property was not converted.	This warning occurs if you attempt to import or convert a Visual Studio setup that has a file search with an invalid value for the MaxDate property. To resolve this warning: **1.** Open the **System Search** view in InstallShield. **2.** Right-click the file search and then click **Modify**. The **System Search Wizard** opens. **3.** On the second panel of this wizard, click the **Details** button. The **File Details** dialog box opens. **4.** Enter the appropriate details.
-9088	InstallShield does not have support for converting Visual Studio Web Setup projects into InstallShield projects.	This error occurs if you attempt to import or convert a Visual Studio Web setup project in InstallShield, since InstallShield does not have support for this conversion. If you encounter this type of error, consider creating a new project in InstallShield and using the Internet Information Services view to manage an IIS Web site on a target system.

Table 12-11 • Visual Studio Project Import Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-9089	InstallShield does not have support for converting Visual Studio CAB projects into InstallShield projects.	This error occurs if you attempt to import or convert a Visual Studio CAB setup project in InstallShield, since InstallShield does not have support for this conversion.
-9090	The Visual Studio project type could not be identified. Therefore, the project was not converted.	This error occurs if you attempt to attempt to import or convert a Visual Studio project type that InstallShield could not identify.
-10000	Conversion canceled by user.	This error occurs if you cancel the conversion process before it finishes.

Upgrade Errors (Upgrading from InstallShield Professional)

This table provides tips for troubleshooting errors that may occur when you upgrade a project (.ism file) created with InstallShield Professional to InstallShield.

Table 12-12 • Tips for Troubleshooting Upgrade Errors

Error Number	Message	Troubleshooting Tips
-61	Could not extract the InstallShield Professional project version. Version 5.5 is assumed.	Make sure the project you are trying to convert is InstallShield Professional version 5.5 or later. If your setup was created using an earlier version of InstallShield Professional, visit the Support Web site for help.
-62	Could not find key LanguageSupport under section [Language] in file <FileName>	The languages your InstallShield Professional setup support could not be determined. You need to define the supported languages manually.
-63	Could not find key ProductName under section [Data] in file <FileName>	The name of your InstallShield Professional project could not be determined. You need to enter this information manually in the Subject field of the Summary Information Stream.
-64	Could not find key Author under section [Data] in file <FileName>	The name of the InstallShield Professional setup could not be determined. You need to manually enter this information in the Author field of the Summary Information Stream.

Table 12-12 • Tips for Troubleshooting Upgrade Errors (cont.)

Error Number	Message	Troubleshooting Tips
-65	Could not find key Version under section [Data] in file <FileName>	The version of your setup project could not be determined. You must manually enter this information in the Product Version setting of the General Information view.
-66	Could not find key HomeUrl under section [Data] in file <FileName>	The URL for your application could not be determined. You need to enter this information in one or all of the following settings in the General Information view: Publisher/Product URL, Support URL, and Product Update URL.
-67	Could not find key CompanyName under section [Data] in file <FileName>	Your Company Name could not be read from the InstallShield Professional project file. You need to manually enter this information in the Publisher setting in the General Information view.
-68	Could not find key InstallationGuid under section [Data] in file <FileName>	The GUID of your installation could not be determined. A new GUID has been created for you in the Product Code setting in the General Information view. If you have the original GUID for your setup, you can overwrite the new GUID generated for you.
-69	Could not find key CurrentFileGroupDef under section [Data] in file <FileName>	A reference to your project's file groups could not be found. Verify your InstallShield Professional's file group settings, save your project, and try upgrading your project again.
-70	Could not find key CurrentComponentDef under section [Data] in file <FileName>	A reference to your project's components could not be found. Verify your InstallShield Professional project's component settings, save your project, and try upgrading your project again.
-79	Could not find key Platforms under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-80	Could not find key Password under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.

Table 12-12 • Tips for Troubleshooting Upgrade Errors (cont.)

Error Number	Message	Troubleshooting Tips
-81	Warning: The Copyright, Department, Email, and Summary settings are not used. Warning: The Descriptions.txt, Instructions.txt, and Notes.txt files are not used. Warning: Use the Source Control option in the Project menu to add your project to source control.	The your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore these warnings.
-82	Could not find key RunDebug under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-241	Could not find key SELFREGISTERING under section <FileGroupName> in file <FileName>	Error determining if your file groups are self-registering. By default, your new components created from this file group are not marked as self-registering. You can change this setting in the property page for the appropriate component.
-242	Could not find key TARGET under section <FileGroupName> in file <FileName>	The Target directory of one of your file groups could not be determined. The component or components created from this file group point to <INSTALLDIR> by default. You can change this setting in the appropriate component's Destination property.
-243	Could not find key LANGUAGE under section <FileGroupName> in file <FileName>	The target languages for one of your file groups could not be determined. The component created from this file group will be language independent. You can change this setting in the appropriate component's Languages property.
-244	Could not find key FOLDER under section [DYNAMIC] in file <FileName>	A reference to your dynamic file links could not be found for one of your file groups. As a result, none of the files referenced by that dynamic link were migrated. You can either open your InstallShield Professional project and verify your dynamic link settings, or manually add these files to your new project.

Table 12-12 • Tips for Troubleshooting Upgrade Errors (cont.)

Error Number	Message	Troubleshooting Tips
-245	Could not find key INCLUDESUBDIR under section [DYNAMIC] in file <FileName>	It could not be determined if subfolders were to be included as part of your dynamic link for one of your file groups. As a result, no subfolders are included in your new project. You can either open your InstallShield Professional project and verify your dynamic link settings, or manually add these files to your new project.
-246	Could not find key WILDCARD under section [DYNAMIC] in file <FileName>	The wildcards you specified for one of your dynamic links could not be read. As a result, all files included in the dynamic link's specified directory were added to your setup. You can either open your InstallShield Professional project and verify your dynamic link settings, or manually remove unwanted files from your new project.
-247	Could not find key FILETYPE under section <SectionName> in file <Path>\Default.fdf	It could not be determined if one of your file groups was marked as shared. As a result, the components created from that file group are not marked as shared. To change this setting, change the appropriate component's Shared property to *Yes*.
-252	Could not find key Folder under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-281	Error opening file <Path>\Default.fdf	A reference to your file groups could not be found. Verify your file group settings in your InstallShield Professional project, save, and open your project in InstallShield 2016 again.
-282	Error Opening file: <FileName>	The upgrade could not locate one of your file groups. Therefore, no files associated with that file group were added to your setup. Verify your file group settings in your InstallShield Professional project, save, and open your project in InstallShield 2016 again.
-283	Could not find section [FILEGROUPS] in file <Path>\Default.fdf	A reference to your file groups could not be found. Verify your file group settings in your InstallShield Professional project, save, and open your project in InstallShield 2016 again.
-284	Could not find section <SectionName> in file <Path>\Default.fdf	One of your file groups could not be found. As a result, none of the files associated with that file group were added to your project. Verify your file group settings in your InstallShield Professional project, save, and open your project in InstallShield 2016 again.

Table 12-12 • Tips for Troubleshooting Upgrade Errors (cont.)

Error Number	Message	Troubleshooting Tips
-285	Could not find key FILETYPE under section <SectionName> in file <Path>\Default.fdf	InstallShield could not determine if the files for one of your file groups was linked to dynamically or statically. As a result, none of that file group's files were added to your setup. Verify your file group settings in your InstallShield Professional project, save, and open your project in InstallShield again.
-286	Could not find section [DYNAMIC] in file <FileName>	InstallShield could not determine if the files for one of your file groups was linked to dynamically or statically. As a result, none of that file group's files were added to your setup. Verify your file group settings in your InstallShield Professional, save, and open your project in InstallShield again.
-301	Could not find key DESCRIPTION under section <SectionName> in file <Path>\Default.cdf	The Description property for one of your components could not be read. Enter this information manually in the Description property for the feature created from your component.
-302	Could not find key DISPLAYTEXT under section <SectionName> in file <Path>\Default.cdf	The display name for one of your components could not be read. Enter this information manually in the **Display Name** property for the feature created from your component.
-303	Could not find key COMMENT under section <SectionName> in file <Path>\Default.cdf	The comments for one of your component could not be found. Enter this information manually in the **Comments** property of the appropriate component.
-306	Could not find key <KeyName> under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-341	Error Opening file: <Path<\Default.cdf	There was an error importing your components. As a result, no features were created. Verify the component settings in your InstallShield Professional, save, and open your project in InstallShield again.
-342	Could not find section [COMPONENTS] in file <Path>\Default.cdf	There was an error importing your components. As a result, no features were created. Verify the component settings in your InstallShield Professional project, save, and open your project in InstallShield again.

Table 12-12 • Tips for Troubleshooting Upgrade Errors (cont.)

Error Number	Message	Troubleshooting Tips
-343	Could not find section <SectionName> in file <Path>\Default.cdf	There was an error importing one of your components. As a result, that component was not migrated. Verify the component settings in your InstallShield Professional project, save, and open your project in InstallShield again.
-401	Error opening file <Path>\Default.reg	The registry information associated with your InstallShield Professional project could not be read. As a result, no registry information was migrated. Verify the registry settings in your InstallShield Professional project, save, and open your project in InstallShield again.
-402	Could not find section [DATA] in file <Path>\Default.re	The registry information associated with your InstallShield Professional project could not be read. As a result, no registry information was migrated. Verify the registry settings in your InstallShield Professional project, save, and open your project in InstallShield again.
-403	Could not find section <SectionName> in file <Path>\Default.re	A portion of your registry information could not be read. As a result, that registry data was not migrated. Verify the registry settings in your InstallShield Professional project, save, and open your project in InstallShield again.
-404	Could not find section <SectionName> in file <Path>\Default.re	A portion of your registry information could not be read. As a result, that registry data was not migrated. Verify the registry settings in your InstallShield Professional project, save, and open your project in InstallShield again.
-421	Could not find key TARGET under section <SectionName> in file <Path>\Default.shl	The target for one of your shortcuts could not be determined. Manually enter this information in the Target property of the appropriate shortcut.
-422	Could not find key ICONFILE under section <SectionName> in file <Path>\Default.shl	The icon for one of your shortcuts could not be determined. Manually enter this information in the Icon File property of the appropriate shortcut.
-423	Could not find key ICONINDEX under section <SectionName> in file <Path>\Default.shl	The icon index for one of your shortcuts could not be determined. Manually enter this information in the Icon Index property of the appropriate shortcut.

Table 12-12 • Tips for Troubleshooting Upgrade Errors (cont.)

Error Number	Message	Troubleshooting Tips
-424	Could not find key DISPLAYTEXT under section <SectionName> in file <Path>\Default.shl	The shortcut text for one of your shortcuts could not be determined. Manually enter this information in the Display Name property of the appropriate shortcut.
-425	Could not find key PARAMETERS under section <SectionName> in file <Path>\Default.shl	The arguments for one of your shortcuts could not be determined. Manually enter this information in the Arguments property of the appropriate shortcut.
-426	Could not find key SHORTCUTKEY under section <SectionName> in file <Path>\Default.shl	The shortcut key for one of your shortcuts could not be determined. Manually enter this information in the Hot Key property of the appropriate shortcut.
-427	Could not find key STARTIN under section <SectionName> in file <Path>\Default.shl	The start in directory for one of your shortcuts could not be determined. Manually enter this information in the Working Directory property for the appropriate shortcut.
-428	Could not find key COMMENTS under section <SectionName> in file <Path>\Default.shl	The comments for one of your shortcuts could not be read. Manually enter this information in the Comments property for the appropriate shortcut.
-429	Could not find key RUN under section <SectionName> in file <Path>\Default.shl	The Run property of one of your shortcuts could not be read. This property will be set to Normal by default in the appropriate shortcut's Run property.
-430	Could not find key Typeunder section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-431	Could not find key Uninstall under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-432	Could not find key DisplayText under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-433	Could not find key Shared under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-434	Could not find key Replace under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.

Table 12-12 • Tips for Troubleshooting Upgrade Errors (cont.)

Error Number	Message	Troubleshooting Tips
-435	Could not find key InternetShortcut under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-440	Could not find key EngineBinding under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-441	Could not find key UpdateMode under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-442	Could not find key UpdateService under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-443	Could not find key LogFile under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.
-461	Error opening file <Path>\Default.shl	The shell object information associated with your InstallShield Professional project could not be read. As a result, no shortcuts were created. Verify that the shell object settings in your InstallShield Professional project, save, and open the project in InstallShield again.
-462	Could not find section [DATA] in file <Path>\Default.shl	The shell object information associated with your InstallShield Professional project could not be read. As a result, no shortcuts were created. Verify that the shell object settings in your InstallShield Professional project, save, and open the project in InstallShield again.
-463	Could not find key <KeyName> under section [DATA] in file <Path>\Default.shl	One of your shortcuts could not be migrated. Manually create this shortcut in the Shortcuts explorer.
-464	Could not find section <SectionName> in file <Path>\Default.shl	One of your shortcuts could not be migrated. Manually create this shortcut in the Shortcuts explorer.
-471	Could not find section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.

Table 12-12 • Tips for Troubleshooting Upgrade Errors (cont.)

Error Number	Message	Troubleshooting Tips
-500	Could not find key <KeyName> under section <SectionName> in file <.ipr file>.	If your project was created with a version of InstallShield Professional earlier than InstallShield Professional 7.0, you can ignore this warning.

Upgrade Warnings (Upgrading from InstallShield Professional)

Because of the differences in architecture between InstallShield Professional and InstallShield 2016, upgrading your installation project might involve some manual adjustment.

If you need to manually adjust settings within your project, a warning appears in the Output window after your project is upgraded. Refer to the associated help topic to learn what you need to do. (cont.)

Table 12-13 • Upgrade Warnings

Warning Number	Message
-289	File link does not exist for a dynamically linked file.
-305	The **Include In Build** property for the InstallShield Professional component could not be directly migrated.
-480	The Shared attribute can be set for only the key file of a component.
-481	The Shared attribute can be set for only the key file of a component. A dynamically linked file cannot be a key file.
-486	Shortcut icons must be linked to at design time.
-487	In InstallShield 2016, installation of files is done according to file versioning rules.
-488	An obsolete event was encountered in your script.
-495	The ODBC core is not installed by default. If you need this functionality in your setup project, navigate to the Redistributables view and select the MDAC merge module.

Upgrade Warning 289: File Link Does Not Exist for Dynamically Linked File

In InstallShield Professional, dynamically linking to your source files allows you to add the contents of complete folders and subfolders to your setup without hard-coding paths or file names. If you add files to the source folder or remove files, the list of included files was dynamically updates when you build your release.

Upgrading to InstallShield 2016

When you upgrade an InstallShield Professional project to InstallShield 2016, and include a file via dynamic file linking that does not exist, Upgrade Warning 289 appears. The warning informs you that a file link has been created, but the file does not exist.

Upgrade Warning -305: Include In Build Property for the InstallShield Professional Component Could Not Be Directly Migrated

If a component in your InstallShield Professional project had its **Include In Build** property set to No, the corresponding feature will have its Release Flags property set to EXCLUDEFROMBUILD. To exclude the feature from your setup, you need to manually set a release flag in the Releases view or the Filtering Settings panel of the Release Wizard.



***Note** • Because the release flag in the **Releases** view indicates which features to include in the setup, the flag you specify in the **Releases** view must be something other than EXCLUDEFROMBUILD.*

Upgrade Warning -480: Incrementing the Shared .dll Reference Count for Files in a File Group

In InstallShield Professional versions 6.x and earlier, setting the shared property for a file group resulted in incrementing the shared .dll reference count for *all* files in that file group on installation. This upgrade migrates this setting to the shared attribute of the component created, in InstallShield 2016, for this file group.

Shared .dll Reference Count Behavior

In InstallShield 2016, setting the shared attribute of a component results in incrementing of the shared .dll reference count of *only* the key file for that component. By definition, there can be only one key file per component. In order to increment the shared reference count for multiple files, each file requires its own component and the file must be the key file in that component.



***Note** • For .exe, .dll, and .ocx files, you can use the Component Wizard to quickly create components for large numbers of files.*

In InstallShield 2016, even if a file is not marked as shared, if a shared .dll reference count already exists for a file, it is automatically incremented by the installation of that file and decremented by the uninstallation of that file.

Additionally, reference counts are maintained for components based on the component GUID (or Component Code property). Therefore, the shared property of a component is useful only where there might be file sharing with legacy installations, or if other components with differing component GUIDs install this file.

Upgrade Warning -481: Incrementing the Shared DLL Reference Count for Dynamically Linked Files in a File Group

In InstallShield Professional version 6.x and previous, setting the shared property for a file group which contained a dynamic file link resulted in incrementing the shared DLL reference count for *all* files in that file group on installation—regardless of the file link type. This upgrade migrates this setting to the shared attribute of the component created, in InstallShield 2016, for this file group.

Shared DLL Reference Count Behavior

In InstallShield 2016, setting the shared attribute of a component results in incrementing of the shared DLL reference count of *only* the key file for that component. Because dynamically linked files cannot be used as key files, the shared reference count is not incremented for any of the component's files.

A component can have only one key file. Therefore, in order to increment the shared reference count for multiple files, each file must be in its own component and the file must be the component's key file. Files must be statically linked if you want the reference count to increment the shared DLL reference count for each file.



***Note** • For EXE, DLL and OCX files, you can use the Component Wizard to quickly create components for large numbers of files.*

In InstallShield 2016—even if a file is not marked as shared—if a shared DLL reference count already exists for a file, it is automatically incremented by the installation of that file and decremented by the uninstallation of that file.

Additionally, reference counts are maintained for components based on the component GUID (or Component Code property). Therefore, the shared property of a component is useful only where there might be file sharing with legacy installations, or if other components with differing component GUIDs install this file.

Upgrade Warning -486: Specifying Shortcut Icons

Shortcut Creation in InstallShield Professional

In InstallShield Professional version 6.x and earlier, shortcut icons were authored from the target machine's perspective. For example, if the shortcut's icon was installed to <TARGETDIR>\MyIcon.ico, that was the qualified path specified for the shortcut icon. At run time, the shortcut creation process would bind to the icon on the target machine.

Shortcut Creation

In InstallShield 2016, shortcut icons must be linked to at design time. Instead of specifying a path on the target machine as the icon file, the project file must specify a path on the build machine as the icon file. The build engine then creates a separate *stream* for your icon in the installation. The installation then uses that stream at run time during its icon creation process. This means that the setup no longer has to install the icon it is using for its shortcuts.

When upgrading your InstallShield Professional projects, InstallShield 2016 attempts to match the target icon path with a file that exists in your setup project. If unable to do so, this warning is displayed.

Correcting the Path to the Shortcut Icon



Tip • *To correct the shortcut icon path:*

1. Go to the **Shortcuts** view in the IDE and click the shortcut specified in the warning message.
2. In the **Icon File** property field, browse to the icon file.



Note • *Shortcut icons are required in InstallShield 2016. If an icon is not specified, the build engine uses the first icon index of the shortcut's target file as the shortcut icon.*

Upgrade Warning -487: Overwriting Files Upon Installation

Installing Files Based on the File Group's Overwrite Property

In InstallShield Professional version 6.x and earlier, you had the ability to set overwrite properties for individual file groups. One file group could install files based on date, while another installed files only if the file version is older than the file version on the target machine.

Installing Files Based on File Versioning Rules

InstallShield 2016 uses a methodology called file versioning rules to determine which files need to be installed on the target machine. The only option at the component level to override file versioning rules is to set the Never Overwrite property to Yes.

File Versioning Alternatives

Although the default file versioning rules are satisfactory for most applications, there might be cases where an application's requirements necessitate that you control the rules used to decide if a file should be overwritten. To this end, there are a couple of alternatives to file versioning that can be used either individually or in conjunction with each other to modify file versioning rules.

REINSTALLMODE

The REINSTALLMODE technique for overriding file versioning rules is a global mechanism for establishing custom versioning rules. Using this technique affects versioning rules for all files in the installation.



Task ***To use the REINSTALLMODE property:***

1. Open the **Property Manager**.
2. Create a property called REINSTALLMODE.
3. Set the value of this property based on the option codes specified for the REINSTALLMODE property.

The installer reads this property at install time and uses its settings to override file versioning. This setting is also used by the installer during a reinstallation or repair of the application.

Companion Files

The companion files technique for overriding file versioning rules enables you to configure a file so that it installs based on the installation of a companion file. This technique is useful when it makes sense to bind the installation states of several files together.

For example, you can bind a non-versioned user data file to a versioned .exe file. If the .exe is not installed, the user data is not installed.

Upgrade Warning -488: Obsolete Event Encountered in Script File

An obsolete event was encountered in your script file. While most InstallShield Professional script events are supported in InstallShield 2016, there are some that are not available.

Events With Comparable Functionality

- **OnInstallingFile**
- **OnUninstallingFile**

These two events have been replaced with the **OnInstallFilesActionBefore** and **OnInstallFilesActionAfter**. These events are called in Install, Uninstall, and Maintenance modes.

The **OnInstallFilesActionBefore** event is called when the installation is preparing to add or remove files from the target machine based on the Action state of the component containing those files. On uninstall, all related actions that require the source file have already been performed—such as removing any shortcuts to the file and unregistering the file's COM data.

The **OnInstallFilesActionAfter** event is called when the installation has finished adding or removing files from the target machine. On Install, the related install actions for a file have yet to be performed, such as the registration of NT Services and creation of ODBC entries.



Note *• When using the* ***OnInstallFilesActionBefore*** *and* ***OnInstallFilesActionAfter*** *events, you can use the Windows Installer API function* ***MsiGetComponentState*** *to determine the action state of a component.*

Event with Partial Support

The **OnSelfRegistrationError** event is called if you are using the **XCopyFile** function to copy your application files to the source media.

Unsupported Events

The following events are not supported in InstallShield 2016:

- **OnFileReadOnly**
- **OnFileLocked**
- **OnNextDisk**
- **OnRemovingSharedFile**
- **OnFileError**

- **OnInternetError**
- **OnMD5Error**

Upgrade Warning 495: ODBC Core Not Installed by Default

The ODBC core is not installed by default. If this is required by your setup project, select the MDAC merge module from the Redistributables view.

MDAC Prerequisite for ODBC Installation

Although InstallShield Professional provided an object that installed the ODBC core as well as ODBC drivers, InstallShield 2016 does not directly support this functionality. Because Microsoft requires you to install ODBC as a subset of MDAC, the Windows Installer does not support direct installation of the ODBC core.

Upgrade Errors and Warnings (Upgrading from InstallShield—Windows Installer Edition)

This table provides tips for troubleshooting errors and warnings that may occur when you upgrade a project (.ism file) created using a version of InstallShield—Windows Installer Edition to InstallShield.

Table 12-14 • Upgrade Errors and Warnings

Error or Warning Number	Message	Troubleshooting Tips
-100	This error occurs if you are upgrading a project that has a registry value set as a keypath for a component.	You cannot set a registry value as a keypath for a component. To avoid this error, remove that keypath from the component and attempt the upgrade again.
-201	This error occurs if your project contains a component with duplicate file names in it. The upgrader keeps only one of these files and deletes the rest.	To avoid this problem, make sure that you do not have duplicate file names within the same component.
-2007 **-2011** **-2121** **-2122**	These errors occur if the upgrader cannot read information from the old table structure, or it cannot write information to the new table structure.	Once the file has been upgraded, you need to navigate to the areas that failed and reinsert your settings.
-6020	The scripting billboard is no longer supported in the Basic MSI project. This property is not being upgraded.	To add support for this property after your project is upgraded, you can convert your project to an InstallScript MSI project and add your billboards in the Support Files/Billboards view.

Table 12-14 • Upgrade Errors and Warnings (cont.)

Error or Warning Number	Message	Troubleshooting Tips
-6021	The path variable cannot contain apostrophes, which are invalid characters. The name is being changed.	To build your project successfully, update all references to this path variable after the project is upgraded.
-6022	Could not open the MSI file specified in a custom action. The source property is not being converted.	To build your project successfully, enter the product code of the MSI file in the custom action's Source property after the project is upgraded.
-6023	The necessary code page is not installed on your system. The associated string table values are not being upgraded correctly.	Install the appropriate code page prior to project upgrade.
-6026	This error occurs if you have a user-defined path variable in your InstallShield—Windows Installer Edition project that is the same as a predefined path variable in InstallShield.	To upgrade successfully, remove the user-defined path variable in the InstallShield—Windows Installer Edition project and upgrade your project to InstallShield. After upgrading, open your project in InstallShield and create a new path variable that is equivalent to the user-defined path variable. Update all references in the InstallShield project.
-6031	The property name "sOldPropertyName" contains invalid characters. The name is being changed to "sNewPropertyName." To build your project successfully, update all references to that property name after your project is upgraded.	After your project upgrades, change all references to the old property name to the new property name.

HRESULT Values for Windows Installer Run-time Errors

ISRegSrv.dll is the InstallShield DLL that is called from custom actions to register and unregister self-registering files on the target system. The DLL displays the Windows Installer run-time error 1904 or 1905 when it fails for particular reasons.

- 1904: Module [2] failed to register. HRESULT [3].
- 1905: Module [2] failed to unregister. HRESULT [3].

[3] could be one of InstallShield's custom HRESULTs.

Table 12-15 • HRESULT Values for Windows Installer Run-time Errors

HRESULT	Description
-2147220475	Invalid file extension as a self-register file.
-2147220474	Executable file extension, but not actually an executable.
-2147220473	Generic registration failure.
-2147220472	Generic unregistration failure.

DIFxAPI Errors (InstallScript Projects)

This table lists DIFxAPI errors that can be returned calling DIFx driver functions. If the return value from a DIFx driver function is a Win32 error (a positive return value), ISERR_WIN_BASE is added to the error so that it is less than ISERR_SUCCESS. For the most updated documentation on Win32 errors, see the MSDN Library.

Table 12-16 • DIFxAPI Errors (InstallScript)

Return Value	Description
ISERR_ISERVICE_NOT_ENABLED	Indicates that DIFx support has not been enabled or was disabled using Disable (SERVICE_DIFX_*). For information on enabling DIFx support, see Installing Device Drivers.
ISERR_WIN_BASE + ERROR_INVALID_FUNCTION	Indicates that the difxapi.dll was not found (in SUPPORTDIR) or that an exported function could not be found. Verify that you have enabled DIFx support in your project.
ERROR_DEPENDENT_APPLICATIONS_EXIST	The function removed an association between the driver package and the specified application but the function did not uninstall the driver package because other applications are associated with the driver package.
ISERR_WIN_BASE + ERROR_ALREADY_EXISTS	The driver package is already preinstalled and was not preinstalled again.
ISERR_WIN_BASE + ERROR_INSUFFICIENT_BUFFER	The pDestInfPath buffer is not large enough to retrieve the requested .inf file path.
ISERR_WIN_BASE + ERROR_FILE_NOT_FOUND	The specified .inf file was not found.
ERROR_DRIVER_PACKAGE_NOT_IN_STORE	An .inf file does not exist in the driver store that corresponds to the .inf file specified by DriverPackageInfPath.

Table 12-16 • DIFxAPI Errors (InstallScript) (cont.)

Return Value	Description
ISERR_WIN_BASE + ERROR_NO_MORE_ITEMS	This error code applies only if the DRIVER_PACKAGE_ONLY_IF_DEVICE_PRESENT constant is specified but the DRIVER_PACKAGE_FORCE constant is not specified. In this case, the function did not preinstall the specified driver package because, although the specified driver package matched devices in the device tree, the driver already installed for each such device is a better match for the device than the specified driver package. This applies to present and nonpresent devices.
ERROR_NO_SUCH_DEVINST	This error code applies only if the DRIVER_PACKAGE_ONLY_IF_DEVICE_PRESENT constant is specified. The function did not preinstall the specified driver package because the driver package does not match a device in the device tree. This applies to present and nonpresent devices.
ERROR_NO_DEVICE_ID	The driver package does not specify a hardware ID or compatible ID that is supported by the current platform.
ERROR_MISSING_FILE	The function did not preinstall the driver package because files referenced by the .inf file could not be found.
ISERR_WIN_BASE + ERROR_CANNOT_MAKE	The function did not preinstall the driver package.
TRUST_E_NOSIGNATURE	The driver package is not signed.
CERT_E_EXPIRED	The certificate used to sign the driver package is expired.
CERT_E_WRONG_USAGE	The certificate for the driver package is not valid for the requested usage. If the driver package does not have a valid WHQL signature, the function returns this error code in the following two situations • In response to a driver signing dialog box, the user chose not to install the driver package. • The DRIVER_PACKAGE_SILENT constant is specified.
CRYPT_E_FILE_ERROR	The catalog file for the driver package was not found, or possibly, some other error occurred when the function attempted to verify the driver package signature.
ERROR_INVALID_CATALOG_DATA	The catalog file for the driver package is not valid or was not found.
ERROR_SIGNATURE_OSATTRIBUTE_MISMATCH	The certificate is not valid for the current Windows version or it is expired.

Table 12-16 • DIFxAPI Errors (InstallScript) (cont.)

Return Value	Description
ISERR_WIN_BASE + ERROR_SHARING_VIOLATION	A component of drive package in the driver store is locked by a thread or process. This can occur, if a process or thread, other than the thread or process being called, is currently accessing the same driver package.
ISERR_WIN_BASE + ERROR_ACCESS_DENIED	Only members of the Administrators group can access this functionality.
ISERR_WIN_BASE + ERROR_BAD_ENVIRONMENT	The Windows version on which this call was made does not support this operation.
ISERR_WIN_BASE + ERROR_INVALID_PARAMETER	A supplied parameter is not valid.
ISERR_WIN_BASE + ERROR_INVALID_NAME	The specified .inf file path is not valid.
ISERR_WIN_BASE + ERROR_FILENAME_EXCED_RANGE	The length, in characters, of the specified .inf file path is greater than the maximum supported length.
ISERR_WIN_BASE + ERROR_CANT_ACCESS_FILE	The driver package files could not be preinstalled because the specified .inf file is in the system .inf file directory.
ISERR_WIN_BASE + ERROR_OUTOFMEMORY	Available system memory was insufficient to preinstall the driver package.
ERROR_UNSUPPORTED_TYPE	The driver package type is not supported.
ERROR_IN_WOW64	The 32 bit version DIFxAPI does not work on Win64 systems. A 64-bit version of DIFxAPI is required.
ERROR_INSTALL_FAILURE	The installation failed.
ISERR_WIN_BASE + ERROR_INVALID_FUNCTION	The driver package is not for a Plug and Play (PnP) function driver.

"String PRODUCT_NAME was not found in string table" Error

While converting an InstallScript MSI project to an InstallScript project, you may encounter this error when your setup project does not run during the conversion. You should verify if *SHELL_OBJECT_FOLDER = @PRODUCT_NAME;* is in your script.



Note • *Setting SHELL_OBJECT_FOLDER in OnFirstUIBefore is not necessary. If you have the following line in your script, you should delete it:*

```
SHELL_OBJECT_FOLDER = @PRODUCT_NAME;"
```

InstallScript Error Information

The InstallScript compiler displays error messages in the Output window. This window, which is normally positioned near the bottom of the main window, has a Compile tab. If the Output window is not present, on the View menu in InstallShield, click Output.

The following types of error messages are displayed to identify errors in your setup script:

Table 12-17 • Error Message Types

Error Message Type	Range
Warnings	C7501 through C7503
Syntax and Compiler Errors	C8001 through C8522
Fatal Errors	F8501 through F8519
Internal Errors	C9001

InstallScript Syntax or Compiler Errors

Syntax errors prevent the file Setup.inx from being created and cause the compiler to display one of the error messages in the list below. A single syntax error can cause the compiler to generate more than one error message. The first error message is triggered by the syntax error. The others are triggered because the compiler is not able to resolve the statement or statements that follow the error.



Tip • *If your script generates more than one syntax error, correct the first one in the list and then recompile. By correcting the first error, you might eliminate one or more of the others.*

Table 12-18 • InstallScript Syntax or Compiler Errors

Error Number	Message
C8001	multiple main programs defined
C8002	function name expected
C8003	function has no prototype declaration
C8004	identifier already declared
C8005	function was declared as DLL function
C8006	missing '(' after function name
C8007	comma expected

Table 12-18 • InstallScript Syntax or Compiler Errors (cont.)

Error Number	Message
C8008	identifier expected
C8009	too many parameters
C8010	missing right parenthesis
C8011	missing 'begin' at start of function
C8012	semicolon expected
C8013	unexpected end of source file
C8014	identifier already defined
C8015	member name already defined
C8016	undefined label:
C8017	expected typedef (struct) name
C8018	typedef object illegal in this context
C8019	expected type declaration
C8020	parameter list expected
C8021	missing 'begin' after 'typedef'
C8022	comma or semicolon expected
C8023	right bracket expected
C8024	invalid array/string size
C8025	undefined identifier
C8026	invalid use of identifier
C8027	missing operand
C8028	unresolved operator
C8031	uncalled function
C8032	missing member reference
C8033	unsubscripted array

Table 12-18 • InstallScript Syntax or Compiler Errors (cont.)

Error Number	Message
C8034	missing expression
C8035	statement label required
C8036	too many arguments
C8037	variable required
C8038	numeric value required
C8039	string value required
C8040	incomplete argument list
C8042	string or array type required
C8043	typedef pointer type required
C8044	typedef type required
C8045	member name not found
C8046	numeric variable required
C8047	can only take address of variable
C8048	macro name missing
C8049	missing expression for #if/#elif
C8050	invalid expression for #if/#elif
C8051	missing end of string literal
C8052	macro expansion text too large
C8053	identifier not a #define macro
C8054	include file name missing
C8055	#elif not preceded by #if
C8057	#else not preceded by #if
C8058	#endif not preceded by #if
C8059	unrecognized preprocessor command

Table 12-18 • InstallScript Syntax or Compiler Errors (cont.)

Error Number	Message
C8062	constant operand required
C8063	input line too long
C8064	unterminated comment
C8065	string literal exceeds 255 characters
C8066	missing #endif statement at end of file
C8067	integer constant too large
C8068	unrecognized character encountered
C8069	preprocessor command must be first on line
C8070	string constant expected
C8071	colon expected
C8072	'elseif' cannot follow 'else'
C8073	missing 'then' keyword
C8074	'else' clause already encountered
C8075	'default' label already encountered
C8076	multiple case labels for statement
C8077	label already defined
C8078	label illegal in this 'for' loop
C8079	invalid statement
C8080	missing arguments for function
C8081	invalid result for assignment
C8082	missing '(' after switch
C8083	missing ')' after switch
C8084	missing 'case' after switch
C8085	missing '='

Table 12-18 • InstallScript Syntax or Compiler Errors (cont.)

Error Number	Message
C8086	missing 'to' or 'downto'
C8087	cannot return value from program
C8088	not inside if statement
C8089	not inside for statement
C8090	not inside repeat statement
C8091	not inside while statement
C8092	function called but not defined
C8093	size required for string in typedef
C8097	syntax error
C8098	missing '.name' after DLL name
C8099	DLL function name expected
C8100	function not defined for this DLL
C8101	must specify DLL for this function
C8112	typedef includes instance of self
C8113	local variables cannot be external
C8114	preprocessor user define error
C8126	'text' : string variable required
C8127	label illegal within try/catch/endcatch
C8128	goto illegal within try/catch/endcatch
C8522	Access is denied.

Error C8001

Message

```
'program' : multiple main programs defined
```

Description

The keyword program was encountered after the main program block.

Troubleshooting Tips

A script can contain only one main program block—beginning with the keyword program and ending with the keyword endprogram. Restructure your script to use only one main program block.

If your script consists of multiple source files, this error message occurs if more than one of those source files contains a main program block.

Error C8002

Message

```
'text' : function name expected
```

Description

The compiler expected to encounter a function name at the location indicated by *text*.

Troubleshooting Information

Check for a missing or mistyped function name before a parameter list.

Check for missing semicolons in preceding lines.

Error C8003

Message

```
'name' : function has no prototype declaration
```

Description

The function specified by *name* has not been declared in a prototype statement.

Troubleshooting Information

If the function has not been declared, insert a prototype statement before the current block (main program or function definition).

If the function has been declared ahead of the current block, compare the function declaration in the prototype statement with the function header in the function definition. Be sure the spelling of the function name is identical in both places and that the number and types of parameters matches.

If the function has been declared later in the script, move the declaration before the current block.

Error C8004

Message

```
'name' : identifier already declared
```

Description

The identifier specified by *name* has already been declared in the current block. Identifiers can be declared only once in the same block.

Troubleshooting Information

Examine the main program block or function block where the error occurred to find the first declaration of the identifier. If the second declaration is simply a duplicate of the first, delete it. If you intended the second declaration to be a different variable than the first, rename the identifier in the second declaration and update all statements in the script that reference it.

If you cannot find a previous declaration of the variable, you may be declaring an identifier whose name is reserved in InstallScript. Reserved words are syntax colored in the script editor.

Error C8005

Message

```
'name' : function was declared as DLL function
```

Description

The function definition for the function specified by *name* is invalid because that function has been declared in a prototype statement as a DLL function.

Troubleshooting Information

If the function is not a DLL function, remove the DLL file name from the function prototype.

If the name of the user-defined function conflicts with the DLL function name, rename the user-defined function and update all statements in the script that reference it. Be sure the user-defined function has been declared in a prototype statement.

Error C8006

Message

```
'text' : missing '(' after function name
```

Description

The compiler expected to find the opening parenthesis that follows a function name in a function definition; the character or characters indicated by *text* were encountered instead.

Troubleshooting Information

Check that the function's parameter list is enclosed within parentheses. If the function takes no parameters, the parentheses will be empty.

Verify that only space characters come between the function name and its parameter list.

Error C8007

Message

```
'text' : comma expected
```

Description

The character or characters specified by *text* were encountered at a location where a comma was expected.

Troubleshooting Information

If the error occurred in a function declaration, then a comma required to separate parameter declarations is probably missing.

If the error occurred in a function call, then a comma required to separate parameters is probably missing.

Error C8008

Message

```
'text' : identifier expected
```

Description

The character or characters specified by *text* were encountered at a location where an identifier was expected.

Troubleshooting Information

If this error occurs in a variable declaration, check that the data type is followed by an identifier, with no intervening punctuation; check that the identifier contains only allowable characters.

If this error occurs in a function call, check that each pair of argument is delimited by one and only one comma.

Error C8009

Message

```
'text' : too many parameters
```

Description

The character or characters specified by *text* were encountered in a function call parameter list that contains more parameters than have been declared for the function.

Troubleshooting Information

To identify the argument or arguments that do not belong in the function call, compare the number and types of arguments in the function call to the number and types of parameters declared for that function.

Error C8010

Message

```
'text' : too many parameters
```

Description

The character or characters specified by *text* were encountered at a location where a right parenthesis was expected.

Troubleshooting Information

This error is usually triggered when the right parentheses is missing from the parameter list of a function header in a function definition.

Error C8011

Message

```
'end' : missing 'begin' at start of function
```

Description

The keyword begin is missing in a function declaration.

Troubleshooting Information

Insert the keyword begin before the first statement in the function. Local variable declarations should appear between the function header and the keyword begin. The function header itself should not be terminated with a semicolon.

Error C8012

Message

```
'text' : semicolon expected
```

Description

The character or characters specified by *text* were encountered at a location where a semicolon was expected.

Troubleshooting Information

Insert a semicolon to terminate the statement that appears just before the error location.

Error C8013

Message

```
unexpected end of source file
```

Description

The compiler reached the end of the source file without encountering the keyword required to close the current program or function block.

Troubleshooting Tips

- If your script contains function definitions, check that each function definition is terminated with the keyword end and a semicolon.
- Check that typedef blocks are closed with an end statement.
- Check that all flow control blocks are correctly closed.
- Check that all #if, #ifdef, and #ifndef statements have a matching #endif statement.

Error C8014

Message

```
'name' : identifier already defined
```

Description

The identifier specified by *name* has already been declared in the structure.

Troubleshooting Information

Examine the script and any scripts included before the error location to find the first declaration of the identifier. If the second declaration is simply a duplicate of the first, delete it. If you intended the second declaration to create a different identifier than the first, rename the identifier in the second declaration and update all statements in the script that reference it.

Error C8015

Message

```
'name' : member name already defined
```

Description

The member identified by *name* has already been declared in the structure.

Troubleshooting Information

Examine the structure in which the error occurred to find the first declaration of the member. If the second declaration is simply a duplicate of the first, delete it. If you intended the second declaration to be a different member than the first, rename the member in the second declaration and update all statements in the script that reference it.

Error C8016

Message

```
'name' : undefined label:
```

Description

The identifier specified by *name* is used in a goto statement to reference a label that has not been defined.

Troubleshooting Information

The error is located in the block directly above the error location. Find the goto statement that references name. If the goto statement is in a program block, it must reference a label that has been defined in the program block. If the goto statement is in a function definition, it must reference a label that has been defined in that function definition. Determine where you want control to flow and define the label at that location.

Error C8017

Message

```
'text' : expected typedef (struct) name
```

Description

The character or characters specified by *text* were encountered at a location where a typedef declaration was expected.

Troubleshooting Information

Check that a data type has been specified for the variable being declared. Verify that the reserved word for that data type is spelled correctly and is in all upper case.

If more than one variable is declared for that data type, check that the individual identifier names are separated by commas and not some other punctuation mark.

Verify that the data declaration is terminated with a semicolon.

If the error occurs in a function prototype that declares a pointer to a structure as a parameter, check that the structure has been declared ahead of the function prototype.

Error C8018

Message

```
'text' : typedef object illegal in this context
```

Description

The structure type specified by *text* was encountered at a location where it is not allowed.

Troubleshooting Information

This error occurs in a function prototype that declares a structure type as one of its parameters. This is not allowed. Instead, declare the parameter as a pointer to a structure by following the name of the structure type with the keyword POINTER, as shown below:

```
typedef RECT
 begin
     SHORT sX;
     SHORT sY;
 end;

 RECT Rectangle;
 RECT POINTER pRect;

 prototype SizeRectangle(RECT POINTER);
```

Error C8019

Message

```
'text' : expected type declaration
```

Description

The character or characters specified by *text* were encountered at a location where a data declaration was expected.

Troubleshooting Information

This error generally occurs when a semicolon is missing at the end of a data declaration in a line ahead of the error location. It also occurs in a function definition when the function header is terminated with a semicolon.

Error C8020

Message

```
'text' : parameter list expected
```

Description

The character or characters specified by *text* were encountered immediately after a prototype declaration for which there was no parameter list.

Troubleshooting Information

Specify the parameter list for the function above the error location. If the function takes no parameters, specify an empty list by placing opening and closing parenthesis after the function name.

Error C8021

Message

```
'text' : missing 'begin' after 'typedef'
```

Description

The character or characters specified by *text* were encountered in a typedef statement where the keyword begin was expected. In a typedef statement, the keyword begin must follow the structure name.

Troubleshooting Information

Insert the keyword begin after the structure name, with no intervening punctuation.

Error C8022

Message

```
'text' : comma or semicolon expected
```

Description

The character or characters specified by *text* were encountered at a location where a comma or semicolon was expected.

Troubleshooting Information

If this error occurs near a data declaration, check that the identifier is valid and that the data declaration is terminated with a semicolon. If two or more identifiers are being declared in the same statement, verify that they are delimited by commas.

Error C8023

Message

```
'text' : right bracket expected
```

Description

The character or characters specified by *text* were encountered at a location where a right bracket was expected.

Troubleshooting Information

This error occurs when the closing right bracket is missing from a reference to an array.

Error C8024

Message

```
'text' : invalid array/string size
```

Description

The character or characters specified by *text* may not be used to declare the size of a string or array.

Troubleshooting Information

If the size indicator is a constant, verify that it is spelled correctly.

Verify that the size of the array is greater than 0.

Error C8025

Message

```
'text' : undefined identifier
```

Description

The identifier specified by *text* has not been declared.

Troubleshooting Information

All identifiers in a script must be declared before they can be referenced in the main program block or in a function block. Check that the identifier has been declared. if it has, check that spelling used in the declaration matches the spelling of the identifier at the error location.



Note *• Your script might be calling an unsupported function. See Unsupported Functions for more information.*

"'INSTALLDIR': undefined identifier" when Compiling After Conversion

Error C8025 specifies that the identifier (INSTALLDIR in this case) has not been declared. All identifiers in a script must be declared before they can be referenced in the main program block or in a function block.



Note *• The exact error message for error C8025 is not a specific error. It is issued to anything undefined.*

As part of the changes that take place during the conversion process (from InstallScript MSI to an InstallScript project), INSTALLDIR should have been changed to TARGETDIR throughout various IDE elements. Error C8025 occurs when the value of TARGETDIR is not set in your script.

Therefore, the value of TARGETDIR must be set in your script, as it is by default in the **OnFirstUIBefore** event handler function.

You can use the following code to set TARGETDIR:

```
if ( ALLUSERS ) then
```

```
    TARGETDIR = PROGRAMFILES ^ IFX_COMPANY_NAME ^ IFX_PRODUCT_NAME;
else
    TARGETDIR = FOLDER_APPDATA ^ IFX_COMPANY_NAME ^ IFX_PRODUCT_NAME;
```

Error C8026

Message

```
'text' : invalid use of identifier
```

Description

The identifier specified by *text* is being used incorrectly in a statement.

Troubleshooting Information

This error message can be triggered by a number of errors, such as the use of a structure type name, rather than the structure name, in an operation. Examine the statement at the error location and check that all identifiers in the statement are being used correctly.

Error C8027

Message

```
missing operand
```

Description

The compiler was unable to find the operand that was expected after an operator.

Troubleshooting Information

This error message may be displayed when the compiler encounters an incomplete statement ending with an operator at the end of the source file.

Error C8028

Message

```
'text' : unresolved operator
```

Description

The character or characters specified by *text* were encountered after an expression that could not be resolved.

Troubleshooting Information

This error occurs when an operand is missing from an expression. Examine each operator in the expression and make sure that each one has the operands it requires.

Error C8031

Message

```
'text' : uncalled function
```

Description

The character or characters specified by *text* were encountered after a function name, where the opening parenthesis of the argument list was expected.

This error occurs in statements in which the result of a function call is assigned to a variable. The message is triggered by an error in the argument list that follows the function name.

Troubleshooting Information

Check that the argument list is delimited from the function name by an opening parenthesis.

If the argument list was not specified, insert it after the function name; when calling a function that takes no parameters, place empty set of opening and closing parentheses after the function name.

Error C8032

Message

```
'text' : missing member reference
```

Description

The character or characters specified by *text* were encountered at a location were a member name was expected.

Troubleshooting Information

The line of code at the error location references a structure rather than a member of a structure. Determine which member of the structure you intended to reference and insert that member name. Remember to use the member operator to delimit the structure name and the member name.

Error C8033

Message

```
'text' : unsubscripted array
```

Description

The identifier that precedes the character or characters specified by *text* is an unsubscripted array variable. InstallScript does not allow references to a complete array structure. You must reference elements in the array individually.

Troubleshooting Information

Determine which element of the array you want to reference and indicate its position in the array with a subscript. The subscript must be placed within square brackets and positioned immediately after the array name.

Error C8034

Message

```
'text' : missing expression
```

Description

The character or characters specified by *text* were encountered at a location were an expression was expected.

Troubleshooting Information

This error generally occurs in an assignment statement when the value to be assigned is missing. Insert an expression to the right of the assignment operator.

If the error occurs in a statement that references a constant, verify that the definition of that constant includes both the constant name and a value.

Error C8035

Message

```
'text' : statement label required
```

Description

The character or characters specified by *text* were encountered after the keyword goto, where a label is required.

Troubleshooting Information

Add a label after the keyword goto or delete the goto statement.

Error C8036

Message

```
'name' : too many arguments
```

Description

The function specified by *name* is being called with too many arguments.

Troubleshooting Information

Compare the arguments in the function call with the parameter list in the declaration for that function. Delete the argument or arguments in the function call that are not defined in the declaration.

Error C8037

Message

```
'text' : variable required
```

Description

The constant, literal value or structure member specified by *text* is being passed as an argument in a parameter position that requires a variable.

Troubleshooting Information

Declare a variable of the type required for the parameter; assign the constant or literal value to that variable; then pass the variable instead of the constant or literal value.

InstallShield does not allow a structure member to be passed by reference. Declare a variable of the same type as the structure member; assign the value of the structure member to the new variable; then pass the variable instead. When the function returns, assign the value of variable to the structure member.

Error C8038

Message

```
'text' : numeric value required
```

Description

The non-numeric argument specified by *text* was passed in a parameter position requiring a numeric argument.

Troubleshooting Information

Make sure all arguments passed are of the correct data types for the parameter positions they occupy. Also make sure no arguments are missing.

Error C8039

Message

```
'value' : string value required
```

Description

The number specified by *value* was encountered in an expression, statement, or function argument that requires string data.

Troubleshooting Information

In a string operation, check that all operands and operators are compatible with string data.

In an assignment statement, verify that the value you are assigning to a string variable is itself a string.

In a function call, be sure that you are not passing non-string data in a parameter position that requires a string.

Error C8040

Message

```
'name' : incomplete argument list
```

Description

One or more arguments are missing in the call to the function specified by *name*.

Troubleshooting Information

To identify the missing argument, compare the number and types of arguments in the function call to the number and types of parameters for that function.

If all of the arguments are present, check that they are delimited by commas.

Error C8042

Message

```
'[' : string or array type required
```

Description

The identifier to the left of the bracket must be a string or array type.

Troubleshooting Information

This error occurs when you specify an index value for a variable that is not a string or array. Examine the identifier to the left of the bracket. If it is not a string or array, delete the index value or declare the variable as a string or array.

Error C8043

Message

```
'text' : typedef pointer type required
```

Description

The character or characters specified by *text* can be used only with a pointer to a structure.

Troubleshooting Information

A common reason for this error is the use of a structure pointer operator between a structure name and a member name. The structure pointer operator can be used only to specify the member of a record that is referenced with a pointer to a structure. Replace the structure pointer operator with a member operator.

Error C8044

Message

```
'text' : typedef type required
```

Description

The character or characters specified by *text* can be used only with a structure that has been declared in a typedef statement.

Troubleshooting Information

A common reason for this error is the use of a member operator between a pointer to a structure and a member name. The member operator can be used only to specify the member of a structure that is referenced with a structure name. Replace the member operator with a structure pointer operator.

Error C8045

Message

```
'name' : member name not found
```

Description

The member specified by *name* does not exist in the structure you are referencing.

Troubleshooting Information

Check the structure definition for the specified member.

Check the spelling of the member name.

Error C8046

Message

```
numeric variable required
```

Description

A non-numeric argument was passed in a parameter position requiring a numeric variable.

Troubleshooting Information

Replace the argument with a numeric variable.

Error C8047

Message

```
'&' : can only take address of variable
```

Description

The compiler encountered an address operation on a constant or literal.

Troubleshooting Information

The address operator (&) can be used only with a variable.

Error C8048

Message

```
macro name missing
```

Description

The #define statement at the error location has no macro name.

Troubleshooting Information

Insert a macro name after the #define preprocessor directive and before the macro value.

Error C8049

Message

```
missing expression for #if/#elif
```

Description

The #if or #elif preprocessor directive is not followed by a valid expression to determine the flow of compilation.

Troubleshooting Information

This error message is triggered if an expression has not been specified after an #if or #elif keyword. It can also be triggered if an invalid expression follows one of those directives.

Error C8050

Message

```
'text' : invalid expression for #if/#elif
```

Description

The character or characters specified by *text* are invalid in the expression following the #if or #elif preprocessor command.

Troubleshooting Information

This error message is triggered if an invalid expression after an #if or #elif keyword.

Error C8051

Message

```
missing end of string literal
```

Description

The string literal in a #define statement has no closing quotation mark.

Troubleshooting Information

Look for a #define statement above the line on which the error was detected. For a string constant, enclose the literal in opening and closing quotation marks. Numeric constants should not be enclosed within quotation marks.

Error C8052

Message

```
macro expansion text too large
```

Description

The text specified for the identifier in a #define statement exceeds 256 characters.

Troubleshooting Information

Either shorten the string specified for that identifier or divide the text into two or more shorter strings and specify each in a #define statement. Then concatenate the strings in your program.

Error C8053

Message

```
'name' : identifier not a #define macro
```

Description

The identifier specified by *name,* which is used in an #undef statement, has not been defined.

Troubleshooting Information

Check the spelling of the identifier specified by *name*.

Error C8054

Message

```
include file name missing
```

Description

The preprocessor directive #include does not specify the name of the file to include.

Troubleshooting Information

Specify the name of a source file you want to include or delete the preprocessor directive.

Error C8055

Message

```
#elif not preceded by #if
```

Description

The compiler encountered an #elif directive that was not preceded by an #if directive.

Troubleshooting Information

Examine the statements that precede the error location and restructure the conditional compile directives to start with an #if directive.

Error C8057

Message

```
#else not preceded by #if
```

Description

The compiler encountered an #else directive that was not preceded by an #if directive.

Troubleshooting Information

Examine the statements that precede the error location and restructure the conditional compile directives to start with an #if directive.

Error C8058

Message

```
#endif not preceded by #if
```

Description

The compiler encountered an #endif directive that was not preceded by an #if directive.

Troubleshooting Information

Examine the statements that precede the error location and restructure the conditional compile directives to start with an #if directive.

Error C8059

Message

```
'text' : unrecognized preprocessor command
```

Description

The character or characters specified by *text* and preceded by the symbol # is not recognized as a preprocessor command.

Troubleshooting Information

Verify that the preprocessor command is valid in an InstallShield script and that it is spelled correctly.

Error C8062

Message

```
'text' : constant operand required
```

Description

The character or characters specified by *text* were encountered in a statement where a constant was expected.

Troubleshooting Information

This error message is triggered in a switch...endswitch block when the keyword case is not followed by a constant or literal.

Error C8063

Message

```
input line too long
```

Description

A source code line exceeds 1,024 characters.

Troubleshooting Information

Divide the line into two or more lines.

Error C8064

Message

```
unterminated comment
```

Description

The comment that begins at the error location is not terminated.

Troubleshooting Information

Find the end of the comment. Insert comment termination characters or correct the comment termination characters if they were entered incorrectly.

Error C8065

Message

```
string literal exceeds 255 characters
```

Description

The string literal at the error location has more than 255 characters.

Troubleshooting Information

Divide the string literal into two or more string literals and concatenate them.

Error C8066

Message

```
missing #endif statement at end of file
```

Description

An #if directive that appears in the script does not have a corresponding #endif directive.

Troubleshooting Information

Examine the preprocessor directives in your script. Make sure that all #if directives have a corresponding #endif directive.

Error C8067

Message

```
'number' : integer constant too large
```

Description

The value specified by *number* is too large for the number variable.

Troubleshooting Information

The valid range of NUMBER data is -2,147,483,648 and +2,147,483,647. If you specify a value in the range +2,147,483,647 to +4,294,967,295, that value will overflow and the result will be a negative number. If the value is greater than 4,294,967,299, this error message will be displayed.

Error C8068

Message

```
'char' : unrecognized character encountered
```

Description

The character specified by *char* was not recognized by the compiler.

Troubleshooting Information

Certain characters, such as "$" and "{," are not recognized by the compiler.

Error C8069

Message

```
'#' : preprocessor command must be first on line
```

Description

The preprocessor directive identified by the symbol # was encountered at a location where it is not allowed. A preprocessor directive must be the first keyword in any line where it appears.

Troubleshooting Information

The symbol # is reserved to identify preprocessor directives. It has no other valid use in InstallScript. Examine the line that produced the error. If the symbol # appears in an identifier, rename that identifier there and everywhere else in the script where it is referenced.

Error C8070

Message

```
'text' : string constant expected
```

Description

The character or characters specified by *text* were encountered in a statement where a string constant was expected.

Troubleshooting Information

This error message is triggered in a switch...endswitch block when the keyword case is not followed by a string constant. In a switch...endswitch block, the constants specified in the case statements must be the same data type as the variable or expression result specified in the switch statement.

Error C8071

Message

```
colon expected
```

Description

The compiler expected a colon in the statement.

Troubleshooting Information

This error occurs in a switch statement when the constant after the keyword case is not followed by a colon. Insert a colon between the constants and the statements for that case.

Error C8072

Message

```
'elseif' : 'elseif' cannot follow 'else'
```

Description

The keyword elseif was encountered after an else statement in an if...endif block.

Troubleshooting Information

The keyword elseif cannot be used in an if...endif block after an else statement. Restructure the if...endif block to avoid this error.

Error C8073

Message

```
'text' : missing 'then' keyword
```

Description

The character or characters specified by text were encountered in an if block where the keyword then was expected.

Troubleshooting Information

Locate the if statement that immediately precedes the error location. Insert the keyword then after the conditional expression that follows the keyword if.

Error C8074

Message

```
'else' : 'else' clause already encountered
```

Description

Two else statements were encountered in an if...endif block.

Troubleshooting Information

An if...endif block can include only one else branch. If you require multiple conditional branching, use an if...endif block that includes elseif statements or a switch...endswitch block instead.

Error C8075

Message

```
'text' : 'default' label already encountered
```

Description

A second default case was encountered in a switch...endswitch statement.

Troubleshooting Information

A switch...endswitch statement can include only one instance of the keyword default to specify a default case. Restructure the switch....endswitch statement to eliminate one of the default cases.

In some versions of InstallShield, this error message is displayed when multiple case labels are encountered in a switch block.

Error C8076

Message

```
'case' : multiple case labels for statement
```

Description

The compiler encountered the keyword case where it expected to find the statement or statements to be executed for the last defined case.

Troubleshooting Information

The case statement that immediately precedes the character or characters specified by *text* is incomplete. It may be missing a terminating semicolon.

In some versions of InstallShield, this error message is displayed when more than one default statement is encountered in a switch block.

Error C8077

Message

```
'name' : label already defined
```

Description

The label indicated by *name* has already been used in the main program block or function.

Troubleshooting Information

Each label in the program block or in a function block must be unique.

Error C8078

Message

```
label illegal in this 'for' loop
```

Description

You cannot define a label within a for statement.

Troubleshooting Tips

Examine the statement at the error location. Modify your code so that a label is no longer defined within the for statement.

Error C8079

Message

```
'text' : invalid statement
```

Description

The statement at the location indicated by *text* specifies a process that is not allowed or is invalid in the current context.

Troubleshooting Tips

Examine the statement at the error location. Verify that the process is valid. Examples of invalid statements are assignment statements with a constant or an arithmetic operation on the left side of the assignment operator and an end statement directly below a label.

If the statement at the error location is valid, examine statement blocks (such as record declarations, control statements and function definitions) in the lines above the error location. Verify that each statement block is terminated with the correct keyword.

Error C8080

Message

```
'text' : missing arguments for function
```

Description

The character or characters specified by *text* were encountered after a function call where an argument list was expected.

Troubleshooting Information

Determine which arguments are required by the function and revise the call to include a complete argument list within parentheses.

Error C8081

Message

```
'text' : invalid result for assignment
```

Description

The character or characters specified by *text* cannot be assigned the value specified by the assignment statement.

Troubleshooting Information

This error occurs when you attempt to assign a value to a structure. InstallScript does not allow direct assignment to a structure. You must assign a value to each member of the structure individually.

Error C8082

Message

```
'text' : missing '(' after switch
```

Description

The character or characters specified by *text* were encountered in a switch...endswitch block at a location where a left parenthesis was expected.

Troubleshooting Information

Examine the switch statement in which this error occurred. The expression that follows the keyword switch must be enclosed within parentheses.

Error C8083

Message

```
'text' : missing ')' after switch
```

Description

The character or characters specified by *text* were encountered in a switch...endswitch block at a location where a right parenthesis was expected.

Troubleshooting Information

Examine the switch statement in which this error occurred. The expression that follows the keyword switch must be enclosed within parentheses.

Error C8084

Message

```
'text' : missing 'case' after switch
```

Description

The character or characters specified by *text* were encountered in a switch...endswitch block at a location where a case statement was expected.

Troubleshooting Information

This error will occur if the switch statement is terminated with a semicolon or is not followed immediately by a case statement.

Error C8085

Message

```
'text' : missing '='
```

Description

The character or characters specified by *text* were encountered in a for...endfor block at a location where an equal sign was expected.

Troubleshooting Information

This error occurs when the equal sign is missing in the expression that follows the keyword for. Examine the for statement at which the error occurred and verify that the expression that follows the keyword for is structured correctly.

Error C8086

Message

```
'text' : missing 'to' or 'downto'
```

Description

The character or characters specified by *text* were encountered in a for...endfor block at a location where an equal sign was expected.

Troubleshooting Information

This error occurs when the keyword to or downto is missing in a for statement.

Error C8087

Message

```
cannot return value from program
```

Description

The main program block contains a return statement. You cannot return a value from a program.

Troubleshooting Information

Remove the return statement from the program block.

Error C8088

Message

```
'endif' : not inside if statement
```

Description

The keyword endif could not be matched to a previous if statement.

Troubleshooting Information

Examine the if...endif block at the error location. Verify that the keyword endif has a corresponding if statement. This error is often triggered after an earlier error in an if...endif block.

Error C8089

Message

```
'endfor' : not inside for statement
```

Description

The keyword endfor could not be matched to a previous for statement.

Troubleshooting Information

Examine the for...endfor block at the error location. Verify that the keyword endfor has a corresponding for statement. This error is often triggered after an earlier error in a for...endfor block.

Error C8090

Message

```
'until' : not inside repeat statement
```

Description

The keyword until could not be matched to a previous repeat statement.

Troubleshooting Information

Examine the repeat...until block at the error location. Verify that the keyword until has a corresponding repeat statement. This error is often triggered after an earlier error in a repeat...until block.

Error C8091

Message

```
'endwhile' : not inside while statement
```

Description

The keyword endwhile could not be matched to a previous while statement.

Troubleshooting Information

Examine the while...endwhile block at the error location. Verify that the keyword endwhile has a corresponding while statement. This error is often triggered after an earlier error in a while...endwhile block.

Error C8092

Message

```
'name' : function called but not defined
```

Description

The function specified by *name* has been prototyped and called in the body of the script, but it has not been defined.

Troubleshooting Information

This error can occur if the function name is not spelled identically in the prototype, the function call and the function definition. It can also occur if the function definition appears in a file that should have been included (with an #include statement) but was not.

Error C8093

Message

```
'name' : size required for string in typedef
```

Description

The identifier specified by *name* is a string that has been defined as a member in a typedef block, but it does not include a size specification.

Troubleshooting Information

Specify the size of all STRING declarations in a structure—InstallScript's autosizing feature does not work in typedef statements.

Error C8097

Message

```
'text' :syntax error
```

Description

The character or characters specified by *text* triggered an unclassified syntax error.

Troubleshooting Information

This error message is triggered for syntax errors for which there is not a specific error message.

Error C8098

Message

```
'text' : missing '.name' after DLL name
```

Description

The character or characters specified by *text* were encountered after a DLL file name, where a function in the DLL file was expected.

Troubleshooting Information

When calling a function from a DLL, specify the function name, not the DLL file name.

Error C8099

Message

```
'text' : DLL function name expected
```

Description

The character or characters specified by *text* were encountered after a DLL file name, where a function in the DLL file was expected.

Troubleshooting Information

This error occurs when the function name specified with a DLL file name has not been declared in a prototype statement. Verify that this function has been prototyped; if it has, check the spelling of the function name.

Error C8100

Message

```
'name' : function not defined for this DLL
```

Description

The function specified by *name* has not been prototyped for the DLL whose name precedes it.

Troubleshooting Information

This error occurs when you have prototyped functions from two or more DLLs and then called one of those functions with the wrong DLL name.

Error C8101

Message

```
'name' : must specify DLL for this function
```

Description

The function specified by *name* was prototyped twice from two different DLLs.

Troubleshooting Information

When calling functions with identical names from two or more DLLs, you must qualify the function name by prefixing it with the DLL name followed by a period, as in the example below:

```
    prototype MYDLL.UsefulFunction(INT, POINTER);
    prototype YOURDLL.UsefulFunction(INT, POINTER);

    export prototype MyFunc(HWND);

function MyFunc(hMSI)
    NUMBER nValue;
```

```
    POINTER psvString;
begin
    MYDLL.UsefulFunction(nValue, psvString);
end;
```

Error C8112

Message

`'name' : typedef includes instance of self`

Description

The data type of the member specified by *name* is invalid. A typedef cannot contain a member that is an instance of itself.

Troubleshooting Information

Restructure your typedefs to avoid this error.

Error C8113

Message

`'external' : local variables cannot be external`

Description

The keyword external was encountered in data declaration within a function block.

Troubleshooting Information

Data that is local to a function cannot be declared external.

Error C8114

Message

`'text' : preprocessor user define error`

Description

This message is triggered by an #error directive.

Error C8115



Project • *This information applies to InstallScript projects.*

Message

```
'function name' : return type mismatch with prototype
```

Description

This message occurs when the return type of a script-defined function (the data type of the value returned by the function) is not the same in the function definition as it is in the function declaration. Note that if you do not explicitly specify a return type, the return type is assumed to be NUMBER.

Error C8126



Project • *This information applies to InstallScript projects.*

Message

```
'text' : string variable required
```

Description

The character or characters specified by *text* were encountered at a location where a string declaration was expected.

Troubleshooting Information

To declare an array other than a character array (that is, a string), use parentheses rather than brackets; for example:

```
NUMBER nArray(3);
```

Error C8127

Message

```
label illegal within try/catch/endcatch
```

Description

You cannot define a label within a try...catch...endcatch statement.

Troubleshooting Tips

Examine the statement at the error location. Modify your code so that a label is no longer defined within the try...catch...endcatch statement.

Error C8128

Message

```
goto illegal within try/catch/endcatch
```

Description

You cannot use a goto statement within a try...catch...endcatch statement.

Troubleshooting Tips

Examine the statement at the error location. Modify your code so that a goto statement is no longer used within the try...catch...endcatch statement.

Error C8522



Project • *This information applies to InstallScript projects.*

Message

```
Access is denied.
```

Description

InstallShield could not open the specified file.

Troubleshooting Information

This error message may occur because the specified file is read-only. In the Windows Explorer, locate the file and check its properties.

InstallScript Fatal Errors

Fatal errors are those that prevent the compilation from continuing. They are caused by missing include files, incorrectly specified file names, disk I/O errors, and insufficient system resources; the compiler also has certain built-in limitations, such as the limit on include files, that will trigger a fatal error if exceeded.

Table 12-19 • InstallScript Fatal Errors

Error	Message
F8501	I/O error on action file
F8502	I/O error on debug file
F8503	Can't open script input file
F8504	Can't open .inx output file
F8505	Can't open .dbg debug file
F8506	out of memory
F8508	too many macro expansions in line

Table 12-19 • InstallScript Fatal Errors (cont.)

Error	Message
F8509	invalid delimiter for include file name
F8510	missing delimiter for include file name
F8511	can't open include file
F8512	include file plus path is too large
F8513	too many nested #include files
F8514	#if statements too deeply nested
F8515	macro expansion buffer overflow
F8516	maximum error count reached
F8517	too many include files
F8518	I/O error on link file
F8519	user define fatal error

Error F8501

Message

```
I/O error on action file
```

Description

An error occurred when InstallShield attempted to access an intermediate file created during compilation.

Troubleshooting Information

Each time you compile your setup, InstallShield creates intermediate files with the following file extensions: .dbg, .ino, .inx and .obs. After compilation, these files remain in the scripts folder. This message is displayed when an error occurs on an attempt to access the action file; that file has the extension .obs). Among the possible causes:

- The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- There are insufficient system resources (file handles and/or memory). Close any other applications that are open; then compile again. If the error occurs on a system with sufficient disk space and system resources, investigate other obvious reasons why file access might fail:
  - The action file may be damaged. Rename it or delete it from the Scripts folder; then compile again.
  - If the action file is in a shared folder on a file server, verify that you have network rights to write to that folder.

- If an action file exists from a previous compile, check that its file attributes have not been changed to read-only, system, or hidden. Check too that the file is not already open by another application or user.

Error F8502

Message

```
I/O error on debug file
```

Description

An error occurred when InstallShield attempted to access an intermediate file created during compilation.

Troubleshooting Information

Each time you compile your setup, InstallShield creates intermediate files with the following file extensions: .dbg, .ino, .inx and .obs. After compilation, these files remain in the scripts folder. This message is displayed when an error occurs on an attempt to access the debug file; that file has the extension .dbg). Among the possible causes:

- The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- There are insufficient system resources (file handles and/or memory). Close any other applications that are open; then compile again. If the error occurs on a system with sufficient disk space and system resources, investigate other obvious reasons why file access might fail:
  - The debug file may be damaged. Rename it or delete it from the Scripts folder; then compile again.
  - If the debug file is in a shared folder on a file server, verify that you have network rights to write to that folder.
  - If a .dbg file exists from a previous compile, check that its file attributes have not been changed to read-only, system, or hidden. Check too that the file is not already open by another application or user.

Error F8503

Message

```
Can't open script input file
```

Description

The compiler was unable to open the script file specified.

Troubleshooting Information

Verify the location and name of the file. You may have misspelled the file name or specified an incorrect path.

Error F8504

Message

```
Can't open .inx output file
```

Description

The compiler was unable to create the file for the compiled script.

Troubleshooting Information

This error is generated if InstallShield cannot create or recreate the .inx file. Under normal circumstances, the following problems can cause this error:

- The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- There are insufficient system resources (file handles and/or memory). Close any other applications that are open; then compile again. If the error occurs on a system with sufficient disk space and system resources, investigate other obvious reasons why file access might fail:
  - If the target file is in a shared folder on a file server, verify that you have network rights to write to that folder.
  - If an .inx file exists from a previous compile, check that its file attributes have not been changed to read-only, system, or hidden. Check too that the file is not already open by another application or user.

Error F8505

Message

```
Can't open .dbg debug file
```

Description

The compiler was unable to create the debug file. This file contains the information required to run the script with the debugger.

Troubleshooting Information

The following problems will cause this error:

- The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- There are insufficient system resources (file handles and/or memory). Close any other applications that are open; then compile again.

If the error occurs on a system with sufficient disk space and system resources, investigate other obvious reasons why file access might fail:

- If the debug file is in a shared folder on a file server, verify that you have network rights to write to that folder.
- If a .dbg file exists from a previous compilation, check that its file attributes have not been changed to read-only, system, or hidden. Check too that the file is not already open by another application or user.

Error F8506

Message

```
Out of memory
```

Description

The compiler was unable to allocate sufficient memory to complete the compilation.

Troubleshooting Information

Close any other applications that are open; then compile again. If the problem persists, restart Windows; then load InstallShield and compile again.

Error F8508

Message

```
too many macro expansions in line
```

Description

The specified line contains too many macros; the maximum number of macros per line is 100.

Troubleshooting Information

Reduce the number of macros on the line by breaking the line into two or more lines.

Error F8509

Message

```
invalid delimiter for include file name
```

Description

The file name specified after the preprocessor directive #include is missing the initial quotation mark or angled bracket (<).

Troubleshooting Information

File names specified with the preprocessor directive #include must be enclosed by quotation marks (“file name”) or angled brackets (<file name>). Insert a quotation mark or opening angled bracket before the file name. Note that if you insert an initial quotation mark, the file name must be followed by a closing quotation mark; if you insert an opening angled bracket, the file name must be followed by a closing angled bracket (>).

Error F8510

Message

```
missing delimiter for include file name
```

Description

The file name specified after the preprocessor directive #include is missing the closing quotation mark or angled bracket (<).

Troubleshooting Information

File names specified with the preprocessor directive #include must be enclosed by quotation marks (“file name”) or angled brackets (<file name>). Insert a quotation mark or closing angled bracket after the file name. Note that if the file name has an opening quotation mark, you must insert a quotation mark after the file name; if the file name has an opening angled bracket, you must insert a closing angled bracket (>).

Error F8511

Message

```
cannot open include file
```

Description

The compiler was unable to find or open a file that has been included in the script.

Troubleshooting Tips

The following problems can cause this error:

- The file name in an #include statement is misspelled.
- A file name in an #include statement is specified without a full path, but the file cannot be found in the source directory.
- You do not have rights to open files on the network drive where the file resides.
- The file is damaged.

Error F8512

Message

```
include file plus path is too large
```

Description

The path specified in an #include statement exceeds the maximum length of 256 characters

Troubleshooting Information

Move the included files to a folder that can be referenced with a path whose length is less than 256 characters.

Error F8513

Message

```
too many nested #include files
```

Description

InstallScript allows nesting of include files up to a maximum of 8 levels. This limit has been exceeded.

Troubleshooting Information

Reduce the number of nested levels.

Error F8514

Message

```
#if statements too deeply nested
```

Description

InstallScript allows nesting of #if statements up to a maximum of 10 levels. This limit has been exceeded.

Troubleshooting Information

Reduce the number of nested levels.

Error F8515

Message

```
macro expansion buffer overflow
```

Description

The expansion of macros in the error line exceeds the maximum of 1024 characters.

Troubleshooting Information

Restructure your code to eliminate this error.

Error F8516

Message

```
maximum error count reached
```

Description

The maximum number of errors specified on the Compile/Link tab of the Settings dialog box has been reached.

Troubleshooting Information

For more information about the Compile/Link tab of the Settings dialog box, see Compile/Link Tab.

Error F8517

Message

```
too many include files
```

Description

The maximum number of include was exceeded.

Troubleshooting Information

A setup script can include no more that 100 source files. Combine two or more source files to bring the total number down to 100.

Error F8518

Message

```
I/O error on link file
```

Description

An error occurred when InstallShield attempted to write to the link file.

Troubleshooting Information

Each time you compile your setup, InstallShield creates intermediate files with the following file extensions: .dbg, .ino, .inx and .obs. After compilation, these files remain in the scripts folder. This message is displayed when an error occurs on an attempt to access the link file (.ino).

This error might be caused by the following:

- The disk is full. Create more available disk space by deleting unwanted files and then recompile.
- The link file is damaged. Rename it or delete it from Scripts folder and then recompile.

Error F8519

Message

```
user define fatal error
```

Description

This message is triggered by an #error directive.

InstallScript Internal Errors

An internal error may occur if syntax errors exist.

C9001 internal error

Error C9001

Message

```
'text' : internal error
```

Description

The character or characters specified by *text* triggered an unclassified syntax error.

Troubleshooting Information

This error message is triggered by syntax errors for which there is not a specific error message.

InstallScript Warnings

The compiler issues warning messages to alert you to irregularities in the script that may result in inefficiencies or run-time errors.



Caution • *Warnings do not interrupt script compilation unless "Warnings as errors" is selected in the Compiler Settings dialog, or if the number of warnings exceeds the maximum specified. The default maximum value is 50.*

Table 12-20 • InstallScript Warnings

Warning Number	Message
C7501	macro redefinition
C7502	string/array size exceeds recommended limit
C7503	function defined but never called
C7505	typedef definition differs from previous

Warning C7501

Message

```
'name' : macro redefinition
```

Description

The identifier specified by *name* has already been used in a #define statement.

Troubleshooting Information

If the second definition is simply a duplicate of the first, delete it. If you intended the second definition to be different from the first, rename the identifier in the second definition and update all statements in the script that reference it.

Warning C7502

Message

```
'size' : string/array size exceeds recommended limit
```

Description

The numeric value specified by *size* indicates the size of a string or array that is larger than InstallScript's recommended maximum.

Troubleshooting Information

To avoid unexpected effects, reduce the size of the array.

Warning C7503

Message

```
'name' : function defined but never called
```

Description

The function specified by *name* is not called in your setup.

Troubleshooting Information

To reduce the size of your setup, remove the unused function from your script.

Warning C7505

Message

```
'name' : typedef definition differs from previous
```

Description

The data structure specified by *name* is defined in multiple library files (.obl files).

Troubleshooting Information

If the definitions are simply duplicates, delete all but one. If you intended the definitions to be different, rename the data structures to have unique names and update all statements in the script that reference them.

Virtualization Conversion Errors and Warnings

When you are converting a Windows Installer package to a virtual application, errors and warnings may occur. Some of the errors and warnings are applicable to any type of package virtualization, and others are specific to the virtualization solution for which you are preparing packages.

Error -9000: Unknown Exception

The following table documents this message:

Table 12-1 • Error -9000: Unknown Exception

Category	Description
Type:	Error
Message:	`An unknown exception occurred.`
Cause:	This is an unexpected internal error.
Resolution:	Perform preliminary investigational steps and then contact InstallShield Technical Support.

Error -9001: Unknown COM

The following table documents this message:

Table 12-2 • Error -9001: Unknown COM

Category	Description
Type:	Error
Message:	`Internal error.`
Resolution:	Contact InstallShield Technical Support.

Error -9002: Error Opening Package

The following table documents this message:

Table 12-3 • Error -9002: Error Opening Package

Category	Description
Type:	Error
Message:	`An error occurred when opening the package.`
Cause:	This is an unexpected internal error when reading the Windows Installer package.

Table 12-3 • Error -9002: Error Opening Package

Category	Description
Resolution:	Check to make sure that the package is accessible to the user. If the error persists and the package is on a network share, copy the package locally (to avoid any network connection issues) and try again. If this does not solve the problem, perform these additional investigational steps and then contact InstallShield Technical Support.

Error -9003: Error Saving Package

The following table documents this message:

Table 12-4 • Error -9003: Error Saving Package

Category	Description
Type:	Error
Message:	`An error occurred when saving the package.`
Cause:	This is an unexpected internal error when trying to save the Citrix profile.
Resolution:	Check to see if the user has proper access to the location the profile is being built to. If this does not solve the problem, perform these additional investigational steps and then contact InstallShield Technical Support.

Error -9004: Process Cancelled By User

The following table documents this message:

Table 12-5 • Error -9004: Process Cancelled By User

Category	Description
Type:	Error
Message:	`Process cancelled by user.`
Cause:	The user clicked the **Cancel** button to stop the build.
Resolution:	Restart the build process.

Error -9005: Error Creating Temporary Folder

The following table documents this message:

Table 12-6 • Error -9005: Error Creating Temporary Folder

Category	Description
Type:	Error
Message:	`An error occurred while creating a temporary folder`
Cause:	You encounter this error when the user does not have permission to write to **C:\TMP**, or the drive is out of disk space.
Resolution:	Obtain write access to **C:\TMP**, and free some disk space on the drive, and then rebuild the profile.

Error -9006: Error Decompressing Package

The following table documents this message:

Table 12-7 • Error -9006: Error Decompressing Package

Category	Description
Type:	Error
Message:	`An error occurred while decompressing the package 'PackageName'.`
Cause:	You encounter this error when the package is a compressed Windows Installer package (**.msi**) and errors were generated when InstallShield attempted to perform an administrative installation to extract the files.
Resolution:	When this error occurred, you should have also received a return error code from Windows Installer. Look up that error code in the Windows Installer Help Library to determine the cause of the problem. If you did not receive a return error code from Windows Installer, this error could have been caused by the package not being authored properly. In the Windows Installer package, check to see if the **AdminExecuteSequence** table was defined. If that table is missing, the package cannot be decompressed.

Error -9007: File With Extension Not Found

The following table documents this message:

Table 12-8 • Error -9007: File With Extension Not Found

Category	Description
Type:	Error
Message:	`No file found with the extension 'ComponentKeyName'.`
Cause:	This is an unexpected error that occurred when file extensions were being processed.
Resolution:	Check to make sure that the executable for the file extension exists and that it is set as the key file in its component.

Error -9008: Error Extracting Icon

The following table documents this message:

Table 12-9 • Error -9008: Error Extracting Icon

Category	Description
Type:	Error
Message:	`An error occurred while extracting the icon 'IconKeyName'`
Cause:	This is an unexpected error that occurred when an icon listed in the **Icon** table was being extracted.
Resolution:	Verify that the **Icon** entry in the **Icon** table is valid. If necessary, replace it with a valid icon.

Error -9009: Unknown Provider

The following table documents this message:

Table 12-10 • Error -9009: Unknown Provider

Category	Description
Type:	Error
Message:	`The specified provider is unknown 'ProviderName'.`
Cause:	This is an unexpected internal error.
Resolution:	Invalid data may have been modified via the Direct Editor causing this error. Delete the **Release** you are building, and then create a new one and rebuild.

Error -9010: Invalid Target File Name

The following table documents this message:

Table 12-11 • Error -9010: Invalid Target File Name

Category	Description
Type:	Error
Message:	`The target file name is invalid. 'FileName'`
Cause:	This is an unexpected internal error.
Resolution:	Invalid data may have been modified via the Direct Editor causing this error. Verify the **Name** field on the Citrix Assistant / ThinApp Assistant **Profile Information** page and make sure the name does not contain any invalid file name characters.

Error -9011: Error Reading MSI Table

The following table documents this message:

Table 12-12 • Error -9011: Error Reading MSI Table

Category	Description
Type:	Error
Message:	`Unexpected error reading MSI table 'TableName'`
Cause:	This is an unexpected error that occurred when the specified Windows Installer table was being processed.
Resolution:	Perform preliminary investigational steps and then contact InstallShield Technical Support.

Error -9012: Unexpected Error in Method

The following table documents this message:

Table 12-13 • Error -9012: Unexpected Error in Method

Category	Description
Type:	Error
Message:	`Unexpected error in method 'MethodName'`
Cause:	This is an unexpected internal error.
Resolution:	Perform preliminary investigational steps and then contact InstallShield Technical Support.

Error -9013: Type Library Not Found

The following table documents this message:

Table 12-14 • Error -9013: Type Library Not Found

Category	Description
Type:	Error
Message:	`Type library not found: 'TypeLibraryName'`
Cause:	You encounter this error when a type library file was not found when trying to extract COM information.
Resolution:	Check to see if the type library file exists in the proper location when building the profile. If this does not resolve the problem, perform these additional investigational steps and then contact InstallShield Technical Support.

Error -9014: ShellExecute Failed

The following table documents this message:

Table 12-15 • Error -9014: ShellExecute Failed

Category	Description
Type:	Error
Message:	`ShellExecute failed: 'CommandLine'`
Cause:	You encounter this error when the specified command line failed to launch a process.
Resolution:	Check to see if the executable file name shown is a valid file and that the user has the proper access rights to run it. If this does not resolve the problem, perform these additional investigational steps and then contact InstallShield Technical Support.

Error -9015: Unable to Determine Full Path for Driver

The following table documents this message:

Table 12-16 • Error -9015: Unable to Determine Full Path for Driver

Category	Description
Type:	Warning
Message:	`Unable to determine the full path for driver 'DriverName'`

Table 12-16 • Error -9015: Unable to Determine Full Path for Driver

Category	Description
Cause:	You encounter this error when a driver referenced in the **ODBCDataSource** table is not being installed by the package.
Resolution:	This error can be resolved in one of two ways: **Editing the Windows Installer Package** **1.** Edit the package in InstallShield using Direct Edit mode. **2.** Navigate to the **ISVirtualPackage** table. **3.** Create an entry as follows to identify the full path of the missing driver: **Name**: **<DriverName> Description** **Value**: **Path to Driver** **Manually Installing the Driver** Install the missing driver on your machine and then rebuild the Citrix profile.

Warning -9016: Contents of Table Ignored

The following table documents this message:

Table 12-17 • Warning -9016: Contents of Table Ignored

Category	Description
Type:	Warning
Message:	`Contents of table 'TableName' will be ignored`
Cause:	This error message identifies a known limitation of Citrix conversion.
Resolution:	If the contents of the table is deemed critical, repackage the application, and then rebuild the Citrix profile.

Warning -9017: .NET 1.x Assembly Not Supported

The following table documents this message:

Table 12-18 • Warning -9017: .NET 1.x Assembly Not Supported

Category	Description
Type:	Warning
Message:	`Assembly 'AssemblyName' is a .NET 1.x assembly and will not be converted correctly. Only .NET 2.0/3.0 assemblies are currently supported. You may wish to repackage this package first.`
Cause:	You encounter this error when attempting to convert a package containing a .NET 1.x assembly. Only .NET 2.0/3.0 assemblies are currently supported.
Resolution:	Repackage the application, and then rebuild the Citrix profile.

Warning -9018: Custom Actions Ignored

The following table documents this message:

Table 12-19 • Warning -9018: Custom Actions Ignored

Category	Description
Type:	Warning
Message:	`Custom action 'CustomActionName' will be ignored.`
Cause:	When converting a Windows Installer package to a Citrix profile, all custom actions are ignored. Any modifications to a target machine that a custom action in this Windows Installer package may create will not be propagated into the Citrix profile.  ***Note*** • *When a custom action that does not modify the system or perform any part of the installation (such as a predefined custom action or a Type 19 custom action) is encountered, no message is generated. If a Type 51 custom action is encountered (which sets a property from a formatted text string), it is automatically resolved. If a Type 35 custom action is encountered, it is resolved only if it is referenced in the* ***Directory*** *table.*

Table 12-19 • Warning -9018: Custom Actions Ignored

Category	Description
Resolution:	The resolution that you should perform depends upon the purpose of the custom action: • If the custom action merely automatically enters a value or makes some other kind of minor modification, you can ignore this warning. • If the custom action does something which could change the behavior of the installation (such as setting a Property), you may need to resolve this issue. To resolve this issue, first attempt to launch the converted package on Citrix XenApp. If you receive any application errors, you need to repackage this application.  ***To repackage a Windows Installer package to capture custom action functionality:*** **1.** Use the Repackaging Wizard to repackage this application. The Repackaging Wizard monitors system changes as an application is installed, and then that data is converted into a Repackager project. **2.** Build the Repackager project to generate a revised Windows Installer package. This new Windows Installer package does not contain any custom actions, but (as a result of being repackaged) it will include the functionality performed by the original custom action.

Warning -9019: Conditionalized Components

The following table documents this message:

Table 12-20 • Warning -9019: Conditionalized Components

Category	Description
Type:	Warning
Message:	`There exist one or more conditionalized components which may not be converted correctly`
Cause:	This warning is generated when attempting to convert conditionalized components because conditions on components are not evaluated. As a result, InstallShield includes the component in the virtual package.
Resolution:	If you do not want to include the component in the virtual package, modify your .msi package or your project file to exclude that component. Then rebuild the virtual application. If you do want the virtual package to contain this component, you can ignore this warning.

Error -9020: Directory With Null Parent

The following table documents this message:

Table 12-21 • Error -9020: Directory With Null Parent

Category	Description
Type:	Error
Message:	`Directory 'DirectoryName' has a null parent and will be ignored.`
Cause:	This error occurs if a directory table entry (other than `TARGETDIR`) is null.
Resolution:	Evaluate the ThinApp application to see if it works. If it does not work properly, you may want to consider repackaging the package.

Error -9021: Unable to Extract COM Data

The following table documents this message:

Table 12-22 • Error -9021: Unable to Extract COM Data

Category	Description
Type:	Error
Message:	`Unable to extract COM data for 'FileName'`
Cause:	This Windows Installer package has an entry in the **TypeLib** or **SelfReg** table that contains COM data that InstallShield cannot convert to application data. Depending upon which file cannot be COM extracted, it is possible that this application will still work properly in Citrix XenApp isolation environment if you repackage this Windows Installer package with COM table mapping turned off. COM data is stored in the Windows Registry. So, if you repackage this Windows Installer package, the capture process will be able to obtain all of this data because it captures all modifications to the Registry and does not have to rely on COM extraction.
Resolution:	To resolve this issue, you need to repackage your Windows Installer package with COM table mapping turned off.

Error -9022: Complus Table

The following table documents this message:

Table 12-23 • Error -9022: Complus Table

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'Complus'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **Complus** table. During the conversion process, the **Complus** table is not read.
Resolution:	The **Complus** table contains information needed to install COM+ applications. While Citrix XenApp supports communicating with COM+ applications, it does not support *installing* COM+ services. Therefore, this application cannot be deployed on Citrix XenApp.

Error -9024: FileSFPCatalog

The following table documents this message:

Table 12-24 • Error -9024: FileSFPCatalog

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'FileSFPCatalog'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **FileSFPCatalog** table. During the conversion process, the **FileSFPCatalog** table is not read.
Resolution:	The **FileSFPCatalog** table associates specified files with the catalog files used by system file protection. If this file is necessary for the function of the application, you need to use Repackager to repackage the application.

Warning -9026: LaunchCondition Table

The following table documents this message:

Table 12-25 • Warning -9026: LaunchCondition Table

Category	Description
Type:	Warning
Message:	`The conversion process does not support data in the MSI table 'LaunchCondition'.`

Table 12-25 • Warning -9026: LaunchCondition Table

Category	Description
Cause:	You encounter this warning when the Windows Installer package that you are converting includes a **LaunchCondition** table. During the conversion process, the **LaunchCondition** table is not read.
Resolution:	The **LaunchCondition** table contains a list of conditions that all must be satisfied for the installation to begin. For example, if an application requires Windows XP to run, Windows XP is listed in the **LaunchCondition** table. Because this table is not read, no check is performed. Therefore, when a user on an operating system other than Windows XP launches this Citrix profile, the application may not function properly. To resolve this issue, perform one of the following tasks: • **Option 1: Set Requirements on the Profile Requirements Page**—If the launch conditions only include operating system, service pack, and language requirements, open this package in the Citrix Assistant, and set those Operating System and Language requirements on the **Profile Requirements** page. Then deploy this application as a Citrix profile. • **Option 2: Determine if Launch Conditions are Met**—Review the launch conditions listed in the table, and determine if the desktops in your enterprise meet those requirements. If all of the desktops meet those requirements, you can deploy this application as a Citrix profile. If the desktops do not meet those requirements (such as having Internet Explorer 6.0 instead of 7.0), upgrade those desktops to meet these requirements, and then deploy this application as a Citrix profile.

Warning -9027: LockPermissions Table

The following table documents this message:

Table 12-26 • Warning -9027: LockPermissions Table

Category	Description
Type	Warning
Message:	`The conversion process does not support data in the MSI table 'LockPermissions'.`
Cause:	You encounter this warning when the Windows Installer package that you are converting includes a **LockPermissions** table. During the conversion process, the **LockPermissions** table is not read.

Table 12-26 • Warning -9027: LockPermissions Table

Category	Description
Resolution:	The **LockPermissions** table is used to secure individual portions of your application (files, registry keys, and created folders) in a locked-down environment. Citrix does not support permissions on files, registry keys, or created folders. You cannot modify permissions on any of the application's ACLs (access control lists). Because users will have full permissions when running this application in the isolation environment, this warning will not result in any problems.

Error -9028: MoveFile Table

The following table documents this message.

Table 12-27 • Error -9028: MoveFile Table

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'MoveFile'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **MoveFile** table. During the conversion process, the **MoveFile** table is not read.
Resolution:	This **MoveFile** table contains a list of files to be moved or copied from a specified source directory to a specified destination directory. Because this table is not read, you need to do one of the following to resolve this issue: • **Option 1: Edit the Windows Installer Package**—Open the Windows Installer package in InstallShield and modify it to eliminate the use of the **MoveFile** table by installing additional files in the specified directories. • **Option 2: Repackage the Application**—Use the Repackaging Wizard to repackage this application, and then build the Repackager project to generate a revised Windows Installer package. • **Option 3: Write a Pre-Launch Script**—Write a pre-launch script that performs the file moving operations identified in the **MoveFile** table upon application launch.

Error -9029: MsiDriverPackages Table

The following table documents this message:

Table 12-28 • Error -9029: MsiDriverPackages Table

Category	Description
Type:	Error

Table 12-28 • Error -9029: MsiDriverPackages Table

Category	Description
Message:	`The conversion process does not support data in the MSI table 'MsiDriverPackages'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **MsiDriverPackages** table. During the conversion process, the **MsiDriverPackages** table is not read.
Resolution:	The **MsiDriverPackages** table includes one record for each driver package component in the application. Citrix XenApp does not support any type of driver. For example, when installing a printer, you can install the printer software within the isolation environment, but not the printer drivers. Therefore, to resolve this issue, you need to install any required drivers outside of the isolation environment on the user desktop machines.

Warning -9030: ODBCTranslator Table

The following table documents this message:

Table 12-29 • Warning -9030: ODBCTranslator Table

Category	Description
Type:	Warning
Message:	`The conversion process does not support data in the MSI table 'ODBCTranslator'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **ODBCTranslator** table. During the conversion process, the **ODBCTranslator** table is not read.
Resolution:	The **ODBCTranslator** table lists the ODBC translators belonging to the installation. ODBC translators translate one form of raw data into another form that can be used with a specific database type. Ignoring the **ODBCTranslator** table should not prevent your application from functioning properly. However, if it does, you need to use Repackager to repackage the application.

Warning -9031: RemoveFile Table

The following table documents this message:

Table 12-30 • Warning -9031: RemoveFile Table

Category	Description
Type:	Warning

Table 12-30 • Warning -9031: RemoveFile Table

Category	Description
Message:	`The conversion process does not support data in the MSI table 'RemoveFile'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **RemoveFile** table. During the conversion process, the **RemoveFile** table is not read. This warning is displayed only if the application installation removes files during install (not uninstall).
Resolution:	The **RemoveFile** table contains a list of files to be removed. If this file removal requirement is just a clean-up step, that does not impact the function of the application, you do not need to address this issue. However, if the existence of the files listed in the **RemoveFile** table prevents the application from functioning, you need to set the isolation option of the files to **Ignore** so that they are not visible to the isolation environment. The **Ignore** option directs the isolation environment to always look for the file on the system (ignoring the one inside the isolation environment).

Warning -9032: RemoveIniFile Table

The following table documents this message:

Table 12-31 • Warning -9032: RemoveIniFile Table

Category	Description
Type:	Warning
Message:	`The conversion process does not support data in the MSI table 'RemoveIniFile'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **RemoveIniFile** table. During the conversion process, the **RemoveIniFile** table is not read.
Resolution:	The **RemoveIniFile** table contains the information an application needs to delete from a **.ini** file. If the removal of this entry is necessary for the function of the application, you need to use Repackager to repackage the application.

Warning -9033: RemoveRegistry Table

The following table documents this message:

Table 12-32 • Warning -9033: RemoveRegistry Table

Category	Description
Type:	Warning
Message:	`The conversion process does not support data in the MSI table 'RemoveRegistry'.`

Table 12-32 • Warning -9033: RemoveRegistry Table

Category	Description
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **RemoveRegistry** table. During the conversion process, the **RemoveRegistry** table is not read.
Resolution:	The **RemoveRegistry** table contains the registry information the application needs to delete from the system registry. If this removal requirement is just a clean-up step, that does not impact the function of the application, you do not need to address this issue. However, if the existence of the registry keys listed in the **RemoveRegistry** table prevents the application from functioning, you need to set the isolation option of the registry keys to **Ignore** so that they are not visible to the isolation environment. The **Ignore** option directs the isolation environment to always look for the registry key on the system (ignoring the one inside the isolation environment).

Error -9036: ISCEInstall Table

The following table documents this message:

Table 12-33 • Error -9036: ISCEInstall Table

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'ISCEInstall'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **ISCEInstall** table. During the conversion process, the **ISCEInstall** table is not read.
Resolution:	The **ISCEInstall** table is used to install Windows mobile applications. The conversion of mobile applications to Citrix profiles is not supported.

Error -9037: ISComPlusApplication Table

The following table documents this message:

Table 12-34 • Error -9037: ISComPlusApplication Table

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'ISComPlusApplication'.`

Table 12-34 • Error -9037: ISComPlusApplication Table

Category	Description
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **ISComPlusApplication** table. During the conversion process, the **ISComPlusApplication** table is not read.
Resolution:	The **ISComPlusApplication** table contains information about COM+ applications. While Citrix XenApp supports communicating with COM+ applications, it does not support *installing* COM+ services. Therefore, this application cannot be deployed on Citrix XenApp.

Error -9038: ISPalmApp Table

The following table documents this message:

Table 12-35 • Error -9038: ISPalmApp Table

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'ISPalmApp'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **ISPalmApp** table. During the conversion process, the **ISPalmApp** table is not read.
Resolution:	The **ISPalmApp** table is used to install Palm mobile applications. The conversion of mobile applications to Citrix profiles is not supported.

Error -9039: ISSQLScriptFile Table

The following table documents this message:

Table 12-36 • Error -9039: ISSQLScriptFile Table

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'ISSQLScriptFile'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **ISSQLScriptFile** table. During the conversion process, the **ISSQLScriptFile** table is not read.

Table 12-36 • Error -9039: ISSQLScriptFile Table

Category	Description
Resolution:	The **ISSQLScriptFile** table lists SQL scripts. When a Windows Installer package is installed, it can run an SQL script to update a database. An application running as a Citrix profile cannot update a database. To resolve this issue, you need to update the database prior to using the converted Citrix profile using one of the following methods: • Use scripts to update the database manually. • Update it by running the Windows Installer installation on one of the machines in your network that has access to that database.

Error -9040: ISVRoot Table

The following table documents this message:

Table 12-37 • Error -9040: ISVRoot Table

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'ISVRoot'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **ISVRoot** table. During the conversion process, the **ISVRoot** table is not read.
Resolution:	The **ISVRoot** table installs a Web site. An application running as a Citrix profile in an isolation environment cannot create a Web site. Therefore, creating Citrix profiles for applications that create Web sites during installation is not supported.

Error -9041: ISXmlFile Table

The following table documents this message:

Table 12-38 • Error -9041: ISXmlFile Table

Category	Description
Type:	Error
Message:	`The conversion process does not support data in the MSI table 'ISXmlFile'.`
Cause:	You encounter this error when the Windows Installer package that you are converting includes a **ISXmlFile** table. During the conversion process, the **ISXmlFile** table is not read.

Table 12-38 • Error -9041: ISXmlFile Table

Category	Description
Resolution:	The **ISXmlFile** table modifies XML files. If the modification of XML files is required in order for this application to operate properly, you need to use Repackager to repackage this application.

Error -9051: Package Decompression Canceled

The following table documents this message:

Table 12-39 • Error -9051: Package Decompression Canceled

Category	Description
Type:	Error
Message:	`Package decompression canceled by the user`
Cause:	The user cancelled the process of decompression of compressed MSI packages.

Error -9100: CreateInstance of Package Object Failed

The following table documents this message:

Table 12-40 • Error -9100: CreateInstance of Package Object Failed

Category	Description
Type:	Error
Message:	`CreateInstance of the Citrix package object failed.`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9101: Create Operation of Package Object Failed

The following table documents this message:

Table 12-41 • Error -9101: Create Operation of Package Object Failed

Category	Description
Type:	Error
Message:	`Create operation on Citrix package object failed 'ObjectName'.`

Table 12-41 • Error -9101: Create Operation of Package Object Failed

Category	Description
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9102: Failed to Write Header Information

The following table documents this message:

Table 12-42 • Error -9102: Failed to Write Header Information

Category	Description
Type:	Error
Message:	`Failed to write package header information.`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9103: Citrix Finalization Failed

The following table documents this message:

Table 12-43 • Error -9103: Citrix Finalization Failed

Category	Description
Type:	Error
Message:	`Citrix Finalization Failed`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9104: Citrix Save Failed

The following table documents this message:

Table 12-44 • Error -9104: Citrix Save Failed

Category	Description
Type:	Error
Message:	`Citrix Save Failed`
Cause:	Unexpected internal error. This error may sometimes occur when the profile is to be digitally signed.
Resolution:	Deselect the option to digitally sign the Citrix profile and then rebuild it.

Error -9105: Error Initializing Citrix Writer

The following table documents this message:

Table 12-45 • Error -9105: Error Initializing Citrix Writer

Category	Description
Type:	Error
Message:	`Unexpected error initializing Citrix writer`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9106: Error Initializing Citrix Package

The following table documents this message:

Table 12-46 • Error -9106: Error Initializing Citrix Package

Category	Description
Type:	Error
Message:	`Unexpected error initializing Citrix package`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9107: Error Writing Citrix File Entries

The following table documents this message:

Table 12-47 • Error -9107: Error Writing Citrix File Entries

Category	Description
Type:	Error
Message:	`Unexpected error writing Citrix file entries.`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9108: Error Determining Source File Path

The following table documents this message:

Table 12-48 • Error -9108: Error Determining Source File Path

Category	Description
Type:	Error
Message:	`Unexpected error determining source file path for 'FileName'`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9109: Error Writing Citrix Folder Entries

The following table documents this message:

Table 12-49 • Error -9109: Error Writing Citrix Folder Entries

Category	Description
Type:	Error
Message:	`Unexpected error writing Citrix folder entries`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9110: Error Writing Citrix Registry Entries

The following table documents this message:

Table 12-50 • Error -9110: Error Writing Citrix Registry Entries

Category	Description
Type:	Error
Message:	`Unexpected error writing Citrix registry entries`
Cause:	Unexpected internal error.
Resolution:	First check to see if the product was installed properly. Then, perform preliminary investigational steps and contact InstallShield Technical Support.

Error -9113: Error Writing Citrix INI File Entries

The following table documents this message:

Table 12-51 • Error -9113: Error Writing Citrix INI File Entries

Category	Description
Type:	Error
Message:	`Unexpected error writing Citrix INI file entries`
Cause:	Unexpected internal error.
Resolution:	Perform preliminary investigational steps and then contact InstallShield Technical Support.

Error -9114: Error Writing Citrix Shortcuts

The following table documents this message:

Table 12-52 • Error -9114: Error Writing Citrix Shortcuts

Category	Description
Type:	Error
Message:	`Unexpected error writing Citrix shortcuts`
Cause:	A catastrophic error has occurred while writing shortcuts to the profile.
Resolution:	Verify that shortcuts point to a valid file. Try to narrow down issue by only keeping one shortcut and then try to rebuild.

Error -9115: Error Saving Citrix Profile

The following table documents this message:

Table 12-53 • Error -9115: Error Saving Citrix Profile

Category	Description
Type:	Error
Message:	`Unexpected error saving Citrix profile`
Cause:	A catastrophic error has occurred while saving the Citrix profile.
Resolution:	Perform preliminary investigational steps and then contact InstallShield Technical Support.

Error -9116: Error Creating Empty Citrix Profile

The following table documents this message:

Table 12-54 • Error -9116: Error Creating Empty Citrix Profile

Category	Description
Type:	Error
Message:	`Unexpected error creating empty Citrix profile`
Cause:	InstallShield is unable to create a new internal instance of a Citrix profile.
Resolution:	Perform preliminary investigational steps and then contact InstallShield Technical Support.

Error -9117: Error Creating Intermediate Folder

The following table documents this message:

Table 12-55 • Error -9117: Error Creating Intermediate Folder

Category	Description
Type:	Error
Message:	`Unexpected error creating intermediate folder`
Cause:	InstallShield is unable to create the intermediate folder used for the build. This error could occur if the user does not have permission to write to **C:\TMP**.
Resolution:	Obtain write access to **C:\TMP** and then rebuild the profile.

Error -9118: Error Initializing Citrix Profile

The following table documents this message:

Table 12-56 • Error -9118: Error Initializing Citrix Profile

Category	Description
Type:	Error
Message:	`Unexpected error initializing Citrix profile.`
Cause:	The initial values on the new profile could not be set.
Resolution:	Check the package name, description, version, and security settings for any possible causes.

Error -9119: Error Creating Default Target in Citrix Profile

The following table documents this message:

Table 12-57 • Error -9119: Error Creating Default Target in Citrix Profile

Category	Description
Type:	Error
Message:	`Unexpected error creating default target in Citrix profile`
Cause:	Initial target in the new profile could not be created.
Resolution:	Perform preliminary investigational steps and then contact InstallShield Technical Support.

Error -9120: Error Deleting File From Profile

The following table documents this message:

Table 12-58 • Error -9120: Error Deleting File From Profile

Category	Description
Type:	Error
Message:	`Unexpected error deleting file 'FileName' from profile`
Cause:	Specified file could not be deleted from profile.
Resolution:	Check to see if the file exists and if the user has access rights to the file. If that did not resolve the problem, perform these additional investigational steps and then contact InstallShield Technical Support.

Error -9121: Failed to Copy File into Citrix Profile

The following table documents this message:

Table 12-59 • Error -9121: Failed to Copy File into Citrix Profile

Category	Description
Type:	Error
Message:	`Failed to copy file into Citrix profile. Error: 'Name' File: 'Name'`
Cause:	Specified file could not be copied into profile.
Resolution:	Check to see if the file exists and if the user has access rights to the file. Also, when this error occurred, you should have also received a return error code from Windows Installer. Look up that error code in the Windows Installer Help Library to determine the cause of the problem.

Error -9122: Target Does Not Exist in Citrix Profile

The following table documents this message:

Table 12-60 • Error -9122: Target Does Not Exist in Citrix Profile

Category	Description
Type:	Warning
Message:	`The target for shortcut 'ShortcutName' does not exist in the Citrix profile. Excluding shortcut.`
Cause:	Actual file that shortcut points to is not included in the package.
Resolution:	Exclude the shortcut by clearing the selection on the Citrix Assistant **Profile Shortcuts** page, and then rebuild the profile.

Error -9124: No Shortcuts Created for this Profile

The following table documents this message:

Table 12-61 • Error -9124: No Shortcuts Created for this Profile

Category	Description
Type:	Error
Message:	`No shortcuts were created for this profile`
Cause:	A Citrix profile must include at least one valid shortcut.

Table 12-61 • Error -9124: No Shortcuts Created for this Profile

Category	Description
Resolution:	Add a shortcut on the Citrix Assistant **Profile Shortcuts** page, and then rebuild the profile.

Error -9125: Error Writing Citrix File Type Associations

The following table documents this message:

Table 12-62 • Error -9125: Error Writing Citrix File Type Associations

Category	Description
Type:	Error
Message:	`Unexpected error writing Citrix file type associations`
Cause:	Unable to write file type associations.
Resolution:	Perform preliminary investigational steps and then contact InstallShield Technical Support.

Error -9126: Failed to Sign Profile Using Certificate

The following table documents this message:

Table 12-63 • Error -9126: Failed to Sign Profile Using Certificate

Category	Description
Type:	Error
Message:	`Failed to sign the profile using the supplied certificate`
Cause:	The certificate that is being used in invalid.
Resolution:	Obtain a valid certificate and rebuild the profile.

Error -9127: Could Not Create Script Execution

The following table documents this message:

Table 12-64 • Error -9127: Could Not Create Script Execution

Category	Description
Type:	Error
Message:	`Could not create script execution for 'ScriptName'`

Table 12-64 • Error -9127: Could Not Create Script Execution

Category	Description
Cause:	The specified script contains invalid data.
Resolution:	On the Citrix Assistant **Build Settings** page, delete the script from the profile, re-add it, and then rebuild the profile. If you are still having problems, perform these additional investigational steps and then contact InstallShield Technical Support.

Warning -9128: Duplicate Shortcut

The following table documents this message:

Table 12-65 • Warning -9128: Duplicate Shortcut

Category	Description
Type:	Warning
Message:	`'ShortcutName' shortcut already exists in the profile. Excluding duplicate shortcut.`
Cause:	There are multiple shortcuts defined in this profile that refer to different Start Menu locations or to other locations in the package.
Resolution:	These shortcuts are not needed. On the Citrix Assistant **Profile Shortcuts** page, unselect these shortcuts, and then rebuild the profile.

Warning -9129: Duplicate Shortcut Names

The following table documents this message:

Table 12-66 • Warning -9129: Duplicate Shortcut Names

Category	Description
Type:	Warning
Message:	`'ShortcutName' shortcut already exists in the profile, but with different command line parameters. A new unique shortcut 'NewShortcutName(1)' will be created in the profile.`
Cause:	There are two shortcuts defined in this profile that have the same name, even though they have different command line parameters.
Resolution:	On the Citrix Assistant **Profile Shortcuts** page, rename one of these shortcuts and then rebuild the profile.

Warning -9130: Duplicate Shortcut Targets

The following table documents this message:

Table 12-67 • Warning -9130: Duplicate Shortcut Targets

Category	Description
Type:	Warning
Message:	`'ShortcutName' shortcut already exists in the profile, but with different target. A new unique shortcut 'NewShortcutName(1)' will be created in the profile.`
Cause:	There are two shortcuts defined in this profile that have the same name, even though they have different targets.
Resolution:	On the Citrix Assistant **Profile Shortcuts** page, rename one of these shortcuts and then rebuild the profile.

Warning -9131: Unable to Resolve Installer Variable

The following table documents this message:

Table 12-68 • Warning -9131: Unable to Resolve Installer Variable

Category	Description
Type:	Warning
Message:	`Unable to resolve an installer variable in the string 'StringName'`
Cause:	Not all Windows Installer variables can be resolved at build time. This can result in errors if your application requires a specific value.
Resolution:	Repackage this application and rebuild the profile, or use a constant value in the Windows Installer package.

Warning -9132: 16 Color Shortcut Icon Not Found

The following table documents this message:

Table 12-69 • Warning -9132: 16 Color Shortcut Icon Not Found

Category	Description
Type:	Warning
Message:	`No 16 color icon found for 'ShortcutName' shortcut. Shortcut icon may look distorted when published.`

Table 12-69 • Warning -9132: 16 Color Shortcut Icon Not Found

Category	Description
Cause:	The icon used for this shortcut does not contain a 16-color image. Since Citrix currently does not support images with a higher number of colors, this icon may look distorted when shown and published in Citrix XenApp.
Resolution:	You can modify the shortcut to use a different icon or add a 16-color image to the one currently used.

Warning -9133: Shortcut Icon Not Found

The following table documents this message:

Table 12-70 • Warning -9133: Shortcut Icon Not Found

Category	Description
Type:	Warning
Message:	`No icon found for 'ShortcutName' shortcut. Using generic Windows application icon instead.`
Cause:	If no icon can be loaded for this shortcut, the generic Windows application icon is used. This can happen if the file used is corrupt or if extracting an image from it is not possible.
Resolution:	Modify the shortcut to use a different icon.

Warning -9134: Failure to Extract Icon from Executable

The following table documents this message:

Table 12-71 • Warning -9134: Failure to Extract Icon from Executable

Category	Description
Type:	Warning
Message:	`Failed to extract icon from executable 'filename'. Make sure the executable is not corrupt.`
Cause:	A corrupt icon file may cause this warning.
Resolution:	Modify the shortcut to use a different icon.

Error -9135: Shortcut Target is 16-Bit

The following table documents this message:

Table 12-72 • Error -9135: Shortcut Target is 16-Bit

Category	Description
Type:	Error
Message:	`The target for shortcut 'ShortcutName' is 16-bit. This application may not function properly in the Citrix Isolation Environment.`
Cause:	The file this shortcut points to is a 16-bit application.
Resolution:	Replace file with a newer 32-bit version. Can also test and see if the application works properly in the Citrix environment.

Warning -9136: Some Files May Not Be Decompressed

The following table documents this message:

Table 12-73 • Warning -9136: Some Files May Not Be Decompressed

Category	Description
Type:	Warning
Message:	`Some files may not be decompressed from this package because it contains features with a default install level of 0.`
Cause:	When installing a compressed Windows Installer package, the build engine runs an administrative installation of it to decompress it. One limitation of an administrative installation is that it does not decompress a file if the feature it is contained in has a default install level of **0**. If there are any files in any components contained within those features, InstallShield will generate an error when it attempts to copy those files into the Citrix profile, because they will not exist in the source location.
Resolution:	To resolve this issue, edit the Windows Installer package and set the default install level of that feature to something other than **0**.

Warning -9137: Destination Directory Cannot Be Found

The following table documents this message:

Table 12-74 • Warning -9137: Destination Directory Cannot Be Found

Category	Description
Type:	Warning

Table 12-74 • Warning -9137: Destination Directory Cannot Be Found

Category	Description
Message:	`The destination directory for the 'FileName' file cannot be found. You should consider Repackaging this application before proceeding with the conversion process.`
Cause:	This is an internal error.
Resolution:	Contact InstallShield Technical Support.

Warning -9138: Ignoring a DuplicateFile Table Entry

The following table documents this message:

Table 12-75 • Warning -9138: Ignoring a DuplicateFile Table Entry

Category	Description
Type:	Warning
Message:	`Ignoring a DuplicateFile table entry because unable to resolve the property used for the DestFolder: 'INVALIDPATH'`
Cause:	You might encounter this error when the Windows Installer package that you are converting includes one or more entries in the **DuplicateFile** table, and a property that is used in the DestFolder column for one of those entries in the **DuplicateFile** table cannot be resolved. For example, if the destination for a duplicate file is set by a custom action, that destination cannot be resolved during the conversion.
Resolution:	The **DuplicateFile** table contains a list of files that need to be duplicated during installation, either to a different directory than the original file or to the same directory but with a different name. Because a destination in this table cannot be resolved, you need to do one of the following to resolve this issue: • **Option 1: Edit the Windows Installer Package**—Open the Windows Installer package in InstallShield and modify it to eliminate the use of the problematic entry in the **DuplicateFile** table by including any additional copies of that file. • **Option 2: Repackage the Application**—Use the Repackaging Wizard to repackage this application, and then build the Repackager project to generate a revised Windows Installer package. • **Option 3: Write a Pre-Launch Script**—Write a pre-launch script that—upon application launch—performs the file copy operations for the problematic entry in the **DuplicateFile** table.

Warning -9150: Warning Building Windows Installer Package

The following table documents this message:

Table 12-76 • Warning -9150: Warning Building Windows Installer Package

Category	Description
Type:	Warning
Message:	`Warning building Windows Installer package: Warning number and message.`
Cause:	This warning indicates that InstallShield encountered a build warning when it built the new Basic MSI project and releases for the App-V application that you are creating.
Resolution:	You may need to resolve the warning in the Basic MSI project that InstallShield created in the App-VPackage subfolder when you built the App-V application. However, note that InstallShield overwrites the Basic MSI project each time that you rebuild the App-V application.

Error -9151: Error Building Windows Installer Package

The following table documents this message:

Table 12-77 • Error -9150: Error Building Windows Installer Package

Category	Description
Type:	Error
Message:	`Error building Windows Installer package: Error number and message.`
Cause:	This error indicates that InstallShield encountered a build error when it built the new Basic MSI project and releases for the App-V application that you are creating.
Resolution:	You may need to resolve the error in the Basic MSI project that InstallShield created in the App-VPackage subfolder when you built the App-V application. However, note that InstallShield overwrites the Basic MSI project each time that you rebuild the App-V application.

Error -9200: ThinApp Must Be Installed

The following table documents this message:

Table 12-78 • Error -9200: ThinApp Must Be Installed

Category	Description
Type:	Error
Message:	`A licensed or demo version of ThinApp must be installed on this machine in order to successfully build ThinApp applications. (www.vmware.com)`

Table 12-78 • Error -9200: ThinApp Must Be Installed

Category	Description
Cause:	ThinApp is not installed.
Resolution:	Install ThinApp.

Warning -9201: Extension for Shortcut Files Must Be ".exe"

The following table documents this message:

Table 12-79 • Warning -9201: Extension for Shortcut Files Must Be ".exe"

Category	Description
Type:	Warning
Message:	`The extension for the target for shortcut 'ShortcutName' is not '.exe'. Excluding shortcut.`
Cause:	Shortcuts that do not have a filename extension of **.exe** are excluded.
Resolution:	No action is required.

Error -9202: No Applications Were Created

The following table documents this message:

Table 12-80 • Error -9202: No Applications Were Created

Category	Description
Type:	Error
Message:	`No applications were created.`
Cause:	Either the Windows Installer package had no shortcuts or all shortcuts were excluded.
Resolution:	Make sure that the source Windows Installer **.msi** package has at least one shortcut to an **.exe** file.

Error -9203: ThinApp Tool is Missing

The following table documents this message:

Table 12-81 • Error -9203: ThinApp Tool is Missing

Category	Description
Type:	Error
Message:	`ThinApp: 'ToolName' was not found`
Cause:	One of the ThinApp tools required to build a ThinApp application was not found.
Resolution:	Reinstall ThinApp.

Error -9204: Duplicate Shortcut

The following table documents this message:

Table 12-82 • Error -9204: Duplicate Shortcut

Category	Description
Type:	Warning
Message:	`'ShortcutName' shortcut already exists. Excluding duplicate shortcut.`
Cause:	The source package has two shortcuts that both point to the same **.exe** target.
Resolution:	No action is required.

Error -9205: Identically Named Shortcut Already Exists, But With Different Parameters

The following table documents this message:

Table 12-83 • Error -9205: Identically Named Shortcut Already Exists, But With Different Command Line Parameters

Category	Description
Type:	Warning
Message:	`'ShortcutName' shortcut already exists, but with different command line parameters. A new, unique shortcut will be created.`
Cause:	Two shortcuts in the package differed in arguments only.
Resolution:	No action is required.

Error -9206: Identically Named Shortcut Already Exists, But With a Different Target

The following table documents this message:

Table 12-84 • Error -9206: Identically Named Shortcut Already Exists, But With a Different Target

Category	Description
Type:	Warning
Message:	`'ShortcutName' shortcut already exists, but with a different target. A new, unique shortcut will be created.`
Cause:	Two shortcuts differed in the target pointed to only.
Resolution:	No action is required.

Error -9207: Error During Build Process (vregtool.exe)

The following table documents this message:

Table 12-85 • Error -9207: Error During Build Process (vregtool.exe)

Category	Description
Type:	Error
Message:	`An error occurred during the ThinApp build process (vregtool.exe).`
Cause:	An unexpected error occurred while running the vregtool.exe step of the ThinApp build process.
Resolution:	The cause of this error may be discernible by the progress messages that were displayed just before this error occurred. Also, make sure none of the files/folders in the build folder hierarchy are locked.

Error -9208: Error Occurred During Build Process (vftool.exe)

The following table documents this message:

Table 12-86 • Error -9208: Error Occurred During Build Process (vftool.exe)

Category	Description
Type:	Error
Message:	`An error occurred during the ThinApp build process (vftool.exe)`

Table 12-86 • Error -9208: Error Occurred During Build Process (vftool.exe)

Category	Description
Cause:	An unexpected error occurred while running the **vftool.exe** step of the ThinApp build process.
Resolution:	The cause of this error may be discernible by the progress messages that were displayed just before this error occurred. Also, make sure none of the files/folders in the build folder hierarchy are locked.

Error -9209: Error Occurred During Build Process (tlink.exe)

The following table documents this message:

Table 12-87 • Error -9209: Error Occurred During Build Process (tlink.exe)

Category	Description
Type:	Error
Message:	`An error occurred during the ThinApp build process (tlink.exe)`
Cause:	An unexpected error occurred while running the tlink.exe step of the ThinApp build process.
Resolution:	The cause of this error may be discernible by the progress messages that were displayed just before this error occurred. Also, make sure none of the files/folders of the build folder hierarchy are locked.

Error -9300: Unhandled Exception During AdviseFile Operation

The following table documents this message:

Table 12-88 • Error -9300: Unhandled Exception During AdviseFile Operation

Category	Description
Type:	Error
Message:	`An unhandled exception occurred during the AdviseFile operation for rule 'RuleName'`
Resolution:	Contact InstallShield Technical Support.

Error -9301: Unhandled Exception During AdviseRegistry Operation

The following table documents this message:

Table 12-89 • Error -9301: Unhandled Exception During AdviseRegistry Operation

Category	Description
Type:	Error
Message:	`An unhandled exception occurred during the AdviseRegistry operation for rule 'RuleName'`
Resolution:	Contact InstallShield Technical Support.

Error -9302: Unhandled Exception During Command Action

The following table documents this message:

Table 12-90 • Error -9302: Unhandled Exception During Command Action

Category	Description
Type:	Error
Message:	`An unhandled exception occurred during the command action with the description 'CommandActionName'`
Resolution:	Contact InstallShield Technical Support.

Error -9303: Unhandled Exception During Alter File Action

The following table documents this message:

Table 12-91 • Error -9303: Unhandled Exception During Alter File Action

Category	Description
Type:	Error
Message:	`An unhandled exception occurred during the alter file action with the description 'FileName'`
Resolution:	Contact InstallShield Technical Support.

Error -9304: Unhandled Exception During Alter Registry Action

The following table documents this message:

Table 12-92 • Error -9304: Unhandled Exception During Alter Registry Action

Category	Description
Type:	Error
Message:	`An unhandled exception occurred during the alter registry action with the description 'RegistryName'`
Resolution:	Contact InstallShield Technical Support.

Error -9305: Unhandled Exception During Create Action

The following table documents this message:

Table 12-93 • Error -9305: Unhandled Exception During Create Action

Category	Description
Type:	Error
Message:	`An unhandled exception occurred during the create action with the description 'CreateName'`
Resolution:	Contact InstallShield Technical Support.

Error -9306: Unhandled Exception During Execution of Rules Engine

The following table documents this message:

Table 12-94 • Error -9306: Unhandled Exception During Execution of Rules Engine

Category	Description
Type:	Error
Message:	`An unhandled exception occurred during the execution of the rules engine.`
Resolution:	Contact InstallShield Technical Support.

Error -9401: Error Initializing App-V Writer

The following table documents this message:

Table 12-95 • Error -9401: Error Initializing App-V Writer

Category	Description
Type:	Error
Message:	`Unexpected error initializing App-V writer.`
Resolution:	Contact InstallShield Technical Support.

Error -9402: Error Initializing App-V Package

The following table documents this message:

Table 12-96 • Error -9402: Error Initializing App-V Package

Category	Description
Type:	Error
Message:	`Unexpected error initializing App-V package.`
Resolution:	Contact InstallShield Technical Support.

Error -9403: Error Writing App-V File Entries

The following table documents this message:

Table 12-97 • Error -9403: Error Writing App-V File Entries

Category	Description
Type:	Error
Message:	`Unexpected error writing App-V file entries.`
Resolution:	Contact InstallShield Technical Support.

Error -9404: Error Writing App-V Folder Entries

The following table documents this message:

Table 12-98 • Error -9404: Error Writing App-V Folder Entries

Category	Description
Type:	Error
Message:	`Unexpected error writing App-V folder entries`
Resolution:	Contact InstallShield Technical Support.

Error -9405: Error Writing App-V Registry Entries

The following table documents this message:

Table 12-99 • Error -9405: Error Writing App-V Registry Entries

Category	Description
Type:	Error
Message:	`Unexpected error writing App-V registry entries.`
Resolution:	Contact InstallShield Technical Support.

Error -9406: Error Writing App-V INI File Entries

The following table documents this message:

Table 12-100 • Error -9406: Error Writing App-V INI File Entries

Category	Description
Type:	Error
Message:	`Unexpected error writing App-V INI file entries.`
Resolution:	Contact InstallShield Technical Support.

Error -9407: Error Writing App-V Shortcuts

The following table documents this message:

Table 12-101 • Error -9407: Error Writing App-V Shortcuts

Category	Description
Type:	Error
Message:	`Unexpected error writing App-V shortcuts.`
Resolution:	Contact InstallShield Technical Support.

Error -9408: Error Writing App-V File Type Data

The following table documents this message:

Table 12-102 • Error -9408: Error Writing App-V File Type Data

Category	Description
Type:	Error
Message:	`Unexpected error writing App-V file type data.`
Resolution:	Contact InstallShield Technical Support.

Error -9409: Error Saving App-V Data

The following table documents this message:

Table 12-103 • Error -9409: Error Saving App-V Data

Category	Description
Type:	Error
Message:	`Unexpected error saving App-V data.`
Resolution:	Contact InstallShield Technical Support.

Error -9410: Error Determining Source File Path

The following table documents this message:

Table 12-104 • Error -9410: Error Determining Source File Path

Category	Description
Type:	Error
Message:	`Unexpected error determining source file path for 'FileName'.`
Cause:	The installation location of a file, which is determined by some run time property, cannot be determined by the App-V virtual converter.
Resolution:	Locate the file in InstallShield and provide a known directory.

Error -9411: OSD File Template Could Not Be Extracted

The following table documents this message:

Table 12-105 • Error -9411: OSD File Template Could Not Be Extracted

Category	Description
Type:	Error
Message:	`The Microsoft App-V OSD file template could not be extracted. The OSD file generation will not operate properly.`
Resolution:	Contact InstallShield Technical Support.

Error -9412: OSD File Could Not Be Saved

The following table documents this message:

Table 12-106 • Error -9412: OSD File Could Not Be Saved

Category	Description
Type:	Error
Message:	`The Microsoft App-V OSD file could not be saved. The OSD file generation will not operate properly.`
Resolution:	Contact InstallShield Technical Support.

Error -9413: App-V OSD Save

The following table documents this message:

Table 12-107 • Error -4313: App-V OSD Save

Category	Description
Type:	Error
Message:	`The Microsoft App-V OSD file could not be saved. The OSD file generation will not operate properly.`
Resolution:	Contact InstallShield Technical Support.

Warning -9414: Local App-V Application Specified as a Dependency of the Primary Application

The following table documents this message:

Table 12-108 • Warning -9414: Local App-V Application Specified as a Dependency of the Primary Application

Category	Description
Type:	Warning
Message:	`A local App-V application was specified as a dependency of the primary application. The primary application may not run correctly if it is relocated to a different location.`
Cause:	The user specified a dependent App-V application that is either on the local drive or on a mapped drive. This is determined by examining the HREF attribute of the CODEBASE tag in the dependency application's OSD file.
Resolution:	Dependency applications should be referenced by a portable mechanism using either a non-FILE protocol or by using a network URL.

Error -9415: Dependency Application Was Not Found

The following table documents this message:

Table 12-109 • Error -9415: Dependency Application Was Not Found

Category	Description
Type:	Error
Message:	`Dependency application was not found: 'ApplicationName'.`
Cause:	A specified App-V dependency application file was not found.

Table 12-109 • Error -9415: Dependency Application Was Not Found

Category	Description
Resolution:	Check the path of the specified App-V dependency application.

Warning -9416: Invalid Primary Application Directory

The following table documents this message:

Table 12-110 • Warning -9416: Invalid Primary Application Directory

Category	Description
Type:	Error
Message:	`The specified Primary Application Directory, 'DirectoryName', does not exist.`
Cause:	This may be caused if the directories specified in the Windows Installer package have changed after a valid Primary Application Directory was specified.
Resolution:	Specify a valid Primary Application Directory using the supplied browse folder in InstallShield.

Error -9417: Dependency Application's OSD File Contains an Invalid HREF Value

The following table documents this message:

Table 12-111 • Error -9417: Dependency Application's OSD File Contains an Invalid HREF Value

Category	Description
Type:	Error
Message:	`Dependency application OSD file contains an invalid value for the HREF field of the CODEBASE tag: 'HREF_Field_Value'`
Cause:	The CODEBASE tag of the dependency application's OSD file may have an empty or non-existent HREF attribute.
Resolution:	Make sure that the CODEBASE tag of the dependency application's OSD file has a valid HREF attribute.

Error -9418: Error While Privatizing Side-By-Side Assemblies

The following table documents this message:

Table 12-112 • Error -9418: Error While Privatizing Side-By-Side Assemblies

Category	Description
Type:	Error
Message:	`An error occurred while privatizing Side-By-Side assemblies.`
Cause:	When converting to an App-V package, files installed to the win32 Sxs assembly cache need to be privatized so that the App-V runtime can find them. This error occurs if there was an unexpected failure in that process.
Resolution:	Contact InstallShield Technical Support.

Error -9419: Error Inserting Watermark

The following table documents this message:

Table 12-113 • Error -9419: Error Inserting Watermark

Category	Description
Type:	Error
Message:	`An error has occurred inserting the evaluation watermark into the App-V Package.`
Resolution:	Contact InstallShield Technical Support.

Error -9424: Building an App-V 5.x Package

The following table documents this message:

Table 12-114 • Error -9424: Building an App-V 5.x Package

Category	Description
Type:	Error
Message:	`Skipping conversion to Microsoft App-V 5. The minimum operating system requirement for building App-V 5 packages is Windows 8 or Windows Server 2012.`
Resolution:	Although you can configure settings for an App-V 5.x package on any version of Windows that InstallShield supports, building an App-V 5.x package from within InstallShield requires Windows 8 or Windows Server 2012.

Warning -9500: Shortcut Missing

The following table documents this message:

Table 12-115 • Warning -9500: Shortcut Missing

Category	Description
Type:	Warning
Message:	`The target for shortcut 'FileName' does not exist. Excluding shortcut.`
Cause:	The target file of a shortcut in the project does not exist.
Resolution:	Repackage this application and then rebuild the virtual package.

Error -10000: Process Cancelled By User

The following table documents this message:

Table 12-116 • Error -10000: Process Cancelled By User

Category	Description
Type:	Error
Message:	`Process cancelled by user.`
Cause:	User clicked **Cancel** to cancel the profile conversion process.

Warning -10001: Suite File Missing

The following table documents this message:

Table 12-117 • Warning -10001: Suite File Missing

Category	Description
Type:	Warning
Message:	`The suite MSI file 'FileName' is missing and will be excluded from the conversion.`
Cause:	An MSI file that is part of a suite conversion was not found.
Resolution:	Make sure the input file for the suite conversion process exists.

Warning -10002: Suite File is Duplicate

The following table documents this message:

Table 12-118 • Warning -10002: Suite File is Duplicate

Category	Description
Type:	Warning
Message:	`The suite MSI file 'FileName' appears to be the same as the main MSI file and we will exclude this file from the conversion process.`
Cause:	A suite conversion was attempted where the main Windows Installer file (.msi) and one of the additional Windows Installer files specified were the same.
Resolution:	Specify unique Windows Installer files as part of the suite conversion process.

Warning -10003: Application File Missing

The following table documents this message:

Table 12-119 • Warning -10003: Application File Missing

Category	Description
Type:	Error
Message:	`Application file not found 'ApplicationName'`
Cause:	A file referenced by the installation was not found by the App-V virtual converter. It is likely that the file reference is broken in the installation.
Resolution:	Use InstallShield to locate the file in the installation and either fix the link or delete it.

Warning -10004: INI File Missing

The following table documents this message:

Table 12-120 • Warning: -10004: INI File Missing

Category	Description
Type:	Error
Message:	`INI file not found 'INI_File_Name'.`
Resolution:	Contact InstallShield Technical Support.

Fix 11000: Excluding TCPIP Registry Entries

The following table documents this message:

Table 12-121 • Fix 11000: Excluding TCPIP Registry Entries

Category	Description
Type:	Fix
Message:	`Excluding TCPIP registry entries from the Citrix profile.`
Action:	All TCPIP registry entries will be excluded from the Citrix profile.

Fatal Error 11001: Fail on VMware

The following table documents this message:

Table 12-122 • Fatal Error 11001: Fail on VMware

Category	Description
Type:	Fatal
Message:	`VMware cannot be virtualized.`
Cause:	Conversion will fail when the application being virtualized is VMware.
Action:	This error message is displayed: `VMware cannot be virtualized.`

Warning 11003: Control Panel Applet - Citrix

The following table documents this message:

Table 12-123 • Warning 11003: Control Panel Applet - Citrix

Category	Description
Type:	Warning
Message:	`The Control Panel Applet [AppletName] will not appear in Control Panel.`
Action:	A warning is displayed when the application contains a Control Panel applet.

Fix 11004: Control Panel Applet - ThinApp

The following table documents this message:

Table 12-124 • Fix 11004: Control Panel Applet - ThinApp

Category	Description
Type:	Fix
Message:	`Generating shortcut for the Control Panel Applet located at 'DirectoryPath'`
Action:	A default shortcut is created for ThinApp Control Panel applets.

Fatal Error 11005: QuickTime 7.4.1 Causes Fatal Error

The following table documents this message:

Table 12-125 • Error 11005: QuickTime 7.4.1 Causes Fatal Error

Category	Description
Type:	Fatal Error
Message:	`QuickTime 7.4.1 is known to have errors when running from a virtual package. Use QuickTime 7.4.5 instead.`
Cause:	QuickTime 7.4.1 cannot be virtualized correctly.
Resolution:	Obtain QuickTime 7.4.5 and repeat the conversion process.

Fix 11006: Adobe Distiller Exclude AdobePDFSettings

The following table documents this message:

Table 12-126 • Fix 11006: Adobe Distiller Exclude AdobePDFSettings

Category	Description
Type:	Fix
Message:	`Excluding the registry key Software\Adobe\Acrobat Distiller\AdobePDFSettings. Adobe Distiller will recreate these settings on first use.`
Action:	The AdobePDFSettings registry settings are excluded.

Fix 11007: Exclude URL Shortcut

The following table documents this message:

Table 12-127 • Fix 11007: Adobe Distiller Exclude AdobePDFSettings

Category	Description
Type:	Fix
Message:	`Excluding shortcut to .URL file. App-V does not launch these files properly.`
Action:	The shortcut to the .URL file is excluded.

Steps to Take Before Calling Technical Support

Before contacting InstallShield Technical Support, perform the following steps to attempt to clearly identify the problem you are having:

- **Check package**—To determine if this error is caused by a problem with the specific package you are converting, try to build a Citrix profile of a simple package that contains only one file.
- **Check machine and OS**—To determine if this error is caused by a configuration on a particular machine or operating system, attempt to build this Citrix profile on another machine or operating system.
- **Check individual files**—To determine if this is error limited to a specific item, find out if removing or excluding a particular item will build error free.

Application Features Requiring Pre- or Post-Conversion Actions

Some application features are ignored when creating a Citrix profile. Therefore, some additional pre- or post-conversion actions must be taken in order for the virtual application to work.

The following table lists the application features which require additional, manual conversion steps:

Table 12-128 • Application Features Ignored During Profile Conversion

Windows Installer Feature	Manual Conversion Steps
User-Defined Custom Actions	When converting a Windows Installer package to a Citrix profile, all custom actions are ignored. For user-defined custom actions, a warning message is generated. Any modifications to a target machine that a custom action in this Windows Installer package may create will not be propagated into the Citrix profile. The resolution that you should perform depends upon the purpose of the custom action: ● If the custom action merely automatically enters a value or makes some other kind of minor modification, you can ignore this warning. ● If the custom action does something which could change the behavior of the installation (such as setting a Property), you may need to resolve this issue. To resolve this issue, first attempt to launch the converted package on Citrix XenApp. If you receive any application errors, you need to repackage this application, by performing the following steps.  ***To successfully convert a package with user-defined custom actions:*** **1.** Use the Repackaging Wizard to repackage this application.The Repackaging Wizard monitors system changes as an application is installed, and then that data is converted into a Repackager project. **2.** Build the Repackager project to generate a revised Windows Installer package. This new Windows Installer package does not contain any custom actions, but (as a result of being repackaged) it will include the functionality performed by the original custom action.
Services	Citrix XenApp does not support any type of services. Therefore, to resolve this issue, you need to install any required services outside of the isolation environment on the user desktop machines.  ***To successfully convert a package with services:*** **1.** If you have an application and a service bundled in the same Windows Installer package, you need to create a separate Windows Installer package containing just the service. **2.** Install the service on each of the user desktop machines. The Citrix profile of this application should now be able to run in an isolation environment on machines that already have the service installed.
COM+	While Citrix XenApp supports communicating with COM+ applications, it does not support *installing* COM+ services. Therefore, an application that contains COM+ services cannot be deployed on Citrix XenApp.

Automation Interface

The InstallShield Help Library explains how to use the InstallShield interface to author all elements of your installation and then build a release. For advanced developers, InstallShield exposes a COM interface that allows you to perform many of the same tasks from a program, such as a Visual Basic executable, or a script, such as a VBScript file in Windows Scripting Host. By calling methods, setting properties, accessing collections, and so on, through the automation interface, you can open a project and modify its features and component data in many of the same ways that you would in the InstallShield interface.

The highest-level object in the automation interface is the ISWiProject object. The first thing you must do to access the interface is create this object. Then, you can use its objects, methods, and properties to modify your project.



Note • *The automation interface affects a project only at design time and not run time.*

For information on automation interface support in Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, and Merge Module projects, see the following sections:

- Automation Objects
- Automation Collections

For information on automation interface support in Advanced UI and Suite/Advanced UI projects, see the following sections:

- Automation Objects for Advanced UI and Suite/Advanced UI Projects
- Automation Collections for Advanced UI and Suite/Advanced UI Projects

Automation Objects



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Each InstallShield automation object represents an installation element such as a feature, a component, or a file. This section describes each of the InstallShield automation objects.

The highest-level object in the automation interface is the ISWiProject object. The first thing you must do to access the interface is create this object. Then, you can use its objects, methods, and properties to modify your project.

ISWiProject Object

Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Project-specific differences are noted where appropriate.

ISWiProject contains the interface for opening, closing, and saving an InstallShield project file (.ism or .dim). When InstallShield is installed on your system, you can instantiate a copy of ISWiProject with the following VB and VBScript syntax:

```
Set m_ISWiProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
```

To open an installation project, call the OpenProject method, as follows:

```
' Build path to the .ism file
strFile = "C:\InstallShield 2016 Projects\Test.ism"
m_ISWiProject.OpenProject strFile
```

Important *• If you are using the automation interface on a 64-bit machine, you may need to load the automation interface through a 32-bit executable file. Otherwise, you may encounter an error when you instantiate a copy of ISWiProject, or any other object. For more information, see Using the Automation Interface on a 64-Bit System.*

Build Status Events

The ISWiProject object raises the following status events:

- Public Event ProgressIncrement(ByVal lIncrement As Long, pbCancel As Boolean)
- Public Event ProgressMax(ByVal lMax As Long, pbCancel As Boolean)
- Public Event StatusMessage(ByVal sMessage As String, pbCancel As Boolean)

These events are raised during the build process and provide status updates. You can set pbCancel to stop the build. For sample code that demonstrates how to use these events, see Using Build Status Events.

Many feature properties and attributes are available in the automation interface through this object's methods, properties, and collections. The following table lists the ISWiProject Object members. Remember that some methods, properties, and collections are only available for a particular project type.

Members

Table 12-1 • ISWiProject Object Members

Name	Project	Type	Description
ActiveLanguage	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object	Read-Write Property	Gets or sets the default language of the project. Specify a single language as a language ID. The automation interface uses the default language whenever you get or set a localizable property, such as ISWiShortcut's DisplayName. In InstallShield, localizable properties must use a string entry. If the property has a string identifier, the string that you enter for the property becomes the string entry's value for the default language.
AddAdvancedFile	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Method	Creates an advanced file with the specified name and (optionally) in the specified disk image in the current project.
AddAutomaticUpgradeEntry	Basic MSI, InstallScript MSI	Method	Creates a new ISWiAutomaticUpgradeEntry item and returns a handle to that object.
AddComponent	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Creates an ISWiComponent object in the current project.
AddCustomAction	Basic MSI, InstallScript MSI	Method	Creates an ISWiCustomAction object in the current project.
AddFeature	Basic MSI, InstallScript, InstallScript MSI	Method	Creates an ISWiFeature object in the current project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
AddLanguage	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Adds a language to the current project.
AddPathVariable	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Adds a path variable to the current project.
AddProductConfig	Basic MSI, InstallScript MSI	Method	Creates an ISWiProductConfig object in the current project.
AddProperty	Basic MSI, InstallScript MSI	Method	Creates a new ISWiProperty item and returns a handle to that object.
AddSetupFile	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Method	Creates a support file with the specified name in the current project.
AddSetupType	InstallScript, InstallScript Object	Method	Creates a setup type with the specified name in the current project.
AddSqlServerConnection	Basic MSI, DIM, InstallScript, InstallScript MSI	Method	Adds a new server connection to the current project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
AddUpgradeTableEntry	Basic MSI, InstallScript MSI	Method	Creates a new ISWiUpgradeTableEntry item and returns a handle to that object.
AdminExecuteSequence	Basic MSI, InstallScript MSI	Read-only Property	Returns an ISWiSequence collection representing the Administration Execute sequence in the current project.
AdminUISequence	Basic MSI, InstallScript MSI	Read-only Property	Returns an ISWiSequence collection representing the Administration User Interface sequence in the current project.
AdvtExecuteSequence	Basic MSI, InstallScript MSI	Read-only Property	Returns an ISWiSequence collection representing the Advertisement Execute sequence in the current project.
AdvtUISequence	Basic MSI, InstallScript MSI	Read-only Property	Returns an ISWiSequence collection representing the Advertisement User Interface sequence in the current project. (Note that validation rule ICE78 requires the Advertisement User Interface sequence to be empty. For more information, see ICEs.)
BuildPatchConfiguration	Basic MSI, InstallScript MSI	Method	Calling this method builds the patch configuration identified by the strPatchConfiguration variable. This action has the same effect as right-clicking on a patch configuration in the Patch Design view and then selecting 'Build Patch'.
BuildPCPFile	Basic MSI, InstallScript MSI	Method	Builds a patch package based on a .pcp file.
CloseProject	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Closes an InstallShield installation project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
CompanyName	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Specifies the company name.  ***Project*** • *In Basic MSI and InstallScript MSI projects, this property corresponds with the Publisher setting in the General Information view.* *In InstallScript projects, this property corresponds with the Company Name setting in the General Information view.*
CompanyPhone	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Specifies the phone number that you would like end users to call for technical support. This phone number is displayed on the Support Information dialog of Add or Remove Programs. This property corresponds with the Support Phone Number setting in the General Information view.
CompanyURL	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Specifies the URL that you would like end users to visit for technical support. This URL is used as a hyperlink on the Support Information dialog of Add or Remove Programs. This property corresponds with the Publisher/Product URL setting in the General Information view.
CreatePatch	Basic MSI, InstallScript MSI	Method	 ***Caution*** • *The CreatePatch method is deprecated for new development. It has been replaced with the BuildPCPFile and BuildPatchConfiguration methods.*
CreateProject	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Creates a new project file (.ism or .dim).
DeleteAdvancedFile	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Method	Deletes the specified advanced file from the current project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
DeleteComponent	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Deletes the specified component from the project.
DeleteCustomAction	Basic MSI, InstallScript MSI	Method	Deletes an ISWiCustomAction object in the current project.
DeleteMergeModule	Basic MSI, InstallScript MSI	Method	Deletes a merge module from the current project.
DeletePathVariable	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Deletes a path variable from the current project.
DeleteProductConfig	Basic MSI, InstallScript MSI	Method	Deletes an ISWiProductConfig object from the current project.
DeleteProperty	Basic MSI, InstallScript MSI	Method	Deletes a property from the Property table.
DeleteSetupFile	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Method	Deletes the specified support file from the current project.
DeleteSetupType	InstallScript, InstallScript Object	Method	Deletes the specified setup type from the current project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
DeleteSQLConnection	Basic MSI, DIM, InstallScript, InstallScript MSI	Method	Remove a SQL Server connection from the current project.
DeselectLanguage	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Deselects a language from the current project.
DotNetFrameworkPath	Basic MSI, InstallScript MSI	Read-Write Property	Specifies the path to the Microsoft .NET Framework. This parameter is optional.
EnableSwidtag	Basic MSI	Read-Write Property	Enables or disables the creation of a software identification tag for an installation. Following is sample code that disables tag creation: `pProject.EnableSwidtag = False` For more information about software tagging, see Including a Software Identification Tag for Your Product.
ExportProject	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Causes the project file to use the XML representation, which is better suited for source code integration.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
ExportStrings	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Exports string entries to a text file.
ForceUpgrade	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Upgrades the project from an earlier version of InstallShield to the current version.
GenerateGUID	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Generates a new, unique GUID that is returned as a string.
ImportProject	Basic MSI	Method	Converts an .isv file (a text file kept in a source code integration program) to an InstallShield .ism project file. ***Note*** • *The .isv file format has been deprecated to version 2.x of InstallShield for Windows Installer and versions of InstallShield Developer prior to 8.x.*

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
ImportStrings	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Imports string entries from a text file.
INSTALLDIR	Basic MSI, InstallScript MSI	Read-Write Property	Sets the INSTALLDIR property for the project. A typical value is "[ProgramFilesFolder]CompanyName\ProductName".
InstallExecuteSequence	Basic MSI, InstallScript MSI	Read-only Property	Returns an ISWiSequence collection representing the Installation Execute sequence in the current project.
InstallUISequence	Basic MSI, InstallScript MSI	Read-only Property	Returns an ISWiSequence collection representing the Installation User Interface sequence in the current project.
ISWiAdvancedFiles	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Collection	Contains every advanced file (as an ISWiAdvancedFile object) in your project.
ISWiAutomaticUpgradeEntries	Basic MSI, InstallScript MSI	Collection	Returns a collection of ISWiAutomaticUpgradeEntry items. This collection presents a list of Automatic Upgrade Items as they are defined in the IDE.
ISWiComponents	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Collection	Contains all of the components in this project.
ISWiCustomActions	Basic MSI, InstallScript MSI	Collection	Contains all of the custom actions in this project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
ISWiFeatures	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Collection	Contains all features' names and their components.
ISWiLanguages	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Collection	Contains all of the languages included in the current project.
ISWiPathVariables	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Collection	Contains all of the path variables that are associated with the current project.
ISWiProductConfigs	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Collection	Contains all of the product configurations in this project.
ISWiProperties	Basic MSI, DIM, InstallScript MSI, Merge Module	Collection	Contains all of the Windows Installer properties in this project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
ISWiSetupFiles	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Collection	Contains every support file (as an ISWiSetupFile object) in your project.
ISWiSetupTypes	InstallScript, InstallScript Object	Collection	Contains every setup type (as an ISWiSetupType object) in your project.
ISWiSISProperties	Basic MSI, DIM, InstallScript MSI, Merge Module	Collection	Contains all of the Summary Information Stream properties in this project.
ISWiSQLConnections	Basic MSI, DIM, InstallScript, InstallScript MSI	Collection	Contains all of the SQL Server connections in the current project.
ISWiUpgradeTableEntries	Basic MSI, InstallScript MSI	Collection	Returns a collection of ISWiUpgradeTableEntry items. This collection presents a list of static major upgrade items, as defined in the IDE.
MergeModuleSearchPath	Basic MSI, InstallScript MSI	Read-Write Property	Specifies one or more comma-delimited folders that contain the merge module (.msm) files referenced by your project. ***Note*** • *If you plan to use the ISWiRelease.Build method, you should set this property.*
MinimumTargetDotNetVersion	Basic MSI, InstallScript MSI	Read-Write Property	Specifies the minimum version of the .NET Framework that the installation requires on the target system. This parameter is optional. The default is the latest version of the .NET Framework the InstallShield interface supports.
MinimumTargetMSIVersion	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Specifies the minimum version of Windows Installer that the installation requires on the target system. This parameter is optional. The default is the latest version of Windows Installer that the InstallShield interface supports.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
ModuleID and ModuleIDGUID	Merge Module	Read-Write String Properties	These properties are used only with merge module projects. Every merge module has a record in the **ModuleSignature** table. The **ModuleID** field of the **ModuleSignature** table is composed of two sections: ModuleID.ModuleIDGUID. For example, for a ModuleID value of IDID.019880FB_04F2_4D4B_99C9_9A0A12185229, the ModuleID and ModuleIDGUID properties of the ISWiProject object enable the following: • The ModuleID property gets or sets *IDID*. • The ModuleIDGUID property gets or sets *019880FB_04F2_4D4B_99C9_9A0A12185229*.
OnUpgrade	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the value for the Small/Minor/Upgrade Settings options on the Common tab in the Upgrades view. This property lets you specify whether you want end users to be prompted before proceeding with the upgrade. A **Setup.exe** setup launcher is required for this functionality. Available options are: • **eouPrompt (1)**— Prompt. • **eouUpgrade (2)**—Do not prompt the end user; just install the upgrade. • **eouNoOp (0)**—Disable any prompts. Your installation should detect the upgrade scenario and run the installation in upgrade mode.
OpenProject	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Opens an InstallShield installation project file (.ism or .dim). This method fails if the project is open in the InstallShield user interface.
PackageCode	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the package code as a string GUID. The GUID must be surrounded by braces—{12345678-1234-1234-1234-1234567890AB}, for example.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
ProductCode	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Read-Write Property	Gets or sets the product code as a string GUID. The GUID must be surrounded by braces—{12345678-1234-1234-1234-1234567890AB}, for example. This property is not available for merge module projects.
ProductName	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Gets or sets the product name.
ProductVersion	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Contains the version number of the project as a string.
RemoveFeature	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Method	Deletes a feature from the project.
SaveProject	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Saves an InstallShield installation project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
SelfRegistrationMethod	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Specifies the self-registration method that should be used for files that are designated as self-registering. Valid options are: • **esrISSelfReg (0)**—Use the InstallShield self-registration table (ISSelfReg) for all self-registering files. • **esrSelfReg (1)**—Use the Windows Installer self-registration table (SelfReg) for all self-registering files.
SkipUpgradeValidators	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Enables you to skip the upgrade validators that normally run at the end of the build.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
SwidtagProperty	Basic MSI	Read-Write Property	Gets or sets value for the software identification tag settings in the General Information view. This property takes one of the following options for the first parameter: • **eswtpUniqueId (0)**—This parameter specifies the value for the Unique ID setting in the General Information view. • **eswtpTagCreatorName (2)**—This parameter specifies the value for the Tag Creator setting in the General Information view. • **eswtpTagCreatorRegid (3)**—This parameter specifies the value for the Tag Creator ID setting in the General Information view. • **eswtpSfwCreatorName (4)**—This parameter specifies the value for the Software Creator setting in the General Information view. • **eswtpSfwCreatorRegid (5)**—This parameter specifies the value for the Software Creator ID setting in the General Information view. • **eswtpSfwLicensorName (6)**—This parameter specifies the value for the Software Licensor setting in the General Information view. • **eswtpSfwLicensorRegid (7)**—This parameter specifies the value for the Software Licensor ID setting in the General Information view. • **eswtpEntitlementRequired (8)**—This parameter specifies whether your product requires users to have a corresponding software entitlement in order for software reconciliation to be considered successful. Following is sample code that uses this property: `pProject.SwidtagProperty(eswtpTagCreatorRegid) = "regid.1986-12.com.mycompany"`
UpgradeCode	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the Upgrade Code property as a string GUID. The GUID must be surrounded by braces—{12345678-1234-1234-1234-1234567890AB}, for example. This property is not available for merge module projects.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Project	Type	Description
UseXMLProjectFormat	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Boolean Property	Specifies whether the project file should use the XML file format.

AddAdvancedFile Method



Project • *The AddAdvancedFile method applies to the following project types.*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The AddAdvancedFile method creates an advanced file object with the specified name and (optionally) in the specified disk image in the current project.

Syntax

```
AddAdvancedFile (FileName As String, Optional DiskType As ISWiDiskType = edtDisk1) As ISWiAdvancedFile
```

Parameters

Table 12-2 • AddAdvancedFile Method Parameters

Parameter	Description
FileName	Pass the fully qualified filename for the advanced file that you want to add.
DiskType	This optional parameter specifies the Advanced Files folder in which you want to add the advanced file. Specify one of the following values. • **edtDisk1** (1)—Add the advanced file to the Disk 1 folder. This is the default value for this optional parameter. • **edtLastDisk** (-1)—Add the advanced file to the Last Disk folder. • **edtOther** (0)—Add the advanced file to the Other folder.

Example

The following Visual Basic lines demonstrate this method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pAdvancedFile As ISWiAdvancedFile

Set pAdvancedFile = pProj.AddAdvancedFile ("C:\My Files\MyLastDiskFile.ext", edtLastDisk)

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

AddAutomaticUpgradeEntry Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the AddAutomaticUpgradeEntry method to create a new IswiAutomaticUpgradeEntry item and return a handle to that object.

Syntax

```
AddAutomaticUpgradeEntry()
```

Example

The following Visual Basic lines demonstrate this method:

```
'################################################

'# THIS SAMPLE WILL NOT EXECUTE PROPERLY WITHOUT A VALID PROJECT PATH

'################################################
set pProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProject.OpenProject("C:\Documents and Settings\Testlab\My Documents\MySetups\your project name-
21.ism")


'################################################
'To create a new entry

dim pAutoEntry
set pAutoEntry = pProject.AddAutomaticUpgradeEntry()

pAutoEntry.Name ="AutoEntry"
pAutoEntry.ReleaseFlags = "AutoFlag"
pAutoEntry.UpgradedSetupPath = "AutoPath"

'################################################
'To find and delete an existing entry
```

```
set pAutoEntry = nothing
set pAutoEntry = pProject.IswiAutoUpgradeEntries("AutoEntry")
pAutoEntry.Delete

pProject.SaveProject
pProject.CloseProject
```

Applies To

ISWiProject

AddComponent Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The AddComponent method creates a component object with the specified name in the current project.

When the component is added, a GUID is automatically generated for it (see the GUID property for ISWiComponent). After you have added the component, you can set its other properties and associate it with a feature.

Syntax

```
AddComponent (ComponentKey As String) As ISWiComponent
```

Parameters

Table 12-3 • AddComponent Method Parameters

Parameter	Description
ComponentKey	The component key is the same as the name of the component that is displayed in the Setup Design views. This string must contain only alphanumeric characters, dots (.), or underscores (_), and it must begin with a letter or an underscore. ComponentKey must be unique among your project's component names. Component names are case-sensitive.

Example

The following Visual Basic lines demonstrate this method:

```
Dim pComponent As ISWiComponent
Set pComponent = m_pProject.AddComponent ("New_Component")
```

Applies To

- ISWiProject

AddCustomAction Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Call the AddCustomAction method to add a custom action to the current project. To insert a custom action into a sequence, use the InsertCustomAction method.

Syntax

```
AddCustomAction (CustomActionKey As String) as ISWiCustomAction
```

Parameters

Table 12-4 • AddCustomAction Method Parameters

Parameter	Description
CustomActionKey	The string name of the custom action you want to create.

Applies To

- ISWiProject

AddFeature Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The AddFeature method creates a feature with the specified name. AddFeature is a member of ISWiProject. The new feature is added to the current project as a top-level feature, but when this method is a member of an ISWiFeature object, the new feature is added as a subfeature of the current feature.

The feature is created with the same default properties that new features have when you add them in InstallShield.

After you have added the feature, you can set its other properties and associate components with it.

Syntax

```
AddFeature (FeatureKey As String) As ISWiFeature
```

Parameters

Table 12-5 • AddFeature Method Parameters

Parameter	Description
FeatureKey	The feature key is the same as the name of the feature that is displayed in the Setup Design and Features views. The name becomes the index for the new ISWiFeature object. This string must contain only alphanumeric characters, dots (.), or underscores (_), and it must begin with a letter or an underscore. FeatureKey must be unique among your project's feature names. Feature names are case-sensitive.

Example

The following Visual Basic lines demonstrate calling AddFeature to add a feature, a subfeature, and then associate an existing component with the subfeature:

```
Dim pProject As ISWiProject

Set pProject = New ISWiProject
pProject.OpenProject "C:\MySetups\SampleApp.ism"

Dim pFeat1, pFeat2 As ISWiFeature
Dim sFeat1Name, sFeat2Name As String

sFeat1Name = "ParentFeature"
sFeat2Name = "SubFeature"

' Add the feature ParentFeature, then SubFeature under ParentFeature
Set pFeat1 = pProject.AddFeature(sFeat1Name)
Set pFeat2 = pFeat1.AddFeature(sFeat2Name)

Dim pComp
Dim sCompName As String

' Add the existing component NewComponent1 to SubFeature
sCompName = "NewComponent1"
Set pComp = pProject.ISWiComponents.Item(sCompName)
pFeat2.AttachComponent pComp

pProject.SaveProject
pProject.CloseProject
```

Applies To

- ISWiProject
- ISWiFeature

AddLanguage Method

Edition • *Support for multilanguage installations is available with InstallShield Premier Edition.*

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the AddLanguage method to add a language to the current project. Using this method is similar to adding a language in the General Information view in InstallShield. You can set the properties of the new language with the ISWiLanguage object.

Syntax

```
AddLanguage (LangId As String)
```

Parameters

Table 12-6 • AddLanguage Method Parameters

Parameter	Description
LangId	Pass the language ID (as a string) that you want to add to the current project. For example: `oProject.AddLanguage "1035"`

Applies To

- ISWiProject

AddPathVariable Method

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the AddPathVariable method to add a path variable to your project. Using this method is similar to adding a path variable in the Path Variables view in InstallShield. You can set the properties of the new path variable with the ISWiPathVariable object.

Important • *Predefined path variables cannot be modified or deleted. If you attempt to do so, exception error 3142 occurs. For more information, see Predefined Path Variables.*

Syntax

```
AddPathVariable (sName As String) As ISWiPathVariable
```

Parameters

Table 12-7 • AddPathVariable Method Parameters

Parameter	Description
sName	Specifies the name of the path variable.

Applies To

- ISWiProject

AddProductConfig Method

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

Call the AddProductConfig method to add a product configuration to your project. Using this method is similar to adding a product configuration with the Release Wizard or the Releases view of the IDE. You can set the properties of the new product configuration with the ISWiProductConfig object.

Syntax

```
AddProductConfig (ProductConfigKey As String) As ISWiProductConfig
```

Parameters

Table 12-8 • AddProductConfig Method Parameters

Parameter	Description
ProductConfigKey	Holds the name of your new product configuration.

Applies To

- ISWiProject

AddProperty Method



Project • *The AddProperty method applies to the following project types.*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The AddProperty method creates a Windows Installer property with the specified name in the current project.

Syntax

```
AddProperty (Name As String) As ISWiProperty
```

Parameters

Table 12-9 • AddProperty Method Parameters

Parameter	Description
Name	Pass the name for the Windows Installer property that you want to add.

Example

The following Visual Basic lines demonstrate the AddProperty method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"

Dim pProperty As ISWiProperty
Set pProperty = pProj.AddProperty ("MYPROP")
pProperty.Value = "MyPropValue"

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

AddSetupFile Method



Project • *The AddSetupFile method applies to the following project types.*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The AddSetupFile method creates a support file with the specified name in the current project.

Syntax

```
AddSetupFile (FileName As String) As ISWiSetupFile
```

Parameters

Table 12-10 • AddSetupFile Method Parameters

Parameter	Description
FileName	Pass the fully qualified file name for the support file that you want to add.

Example

The following Visual Basic lines demonstrate the AddSetupFile method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pSetupFile As ISWiSetupFile

Set pSetupFile = pProj.AddSetupFile ("C:\My Files\MySupportFile.ext")

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

AddSetupType Method



Project • *This information applies to the following project types:*

- *InstallScript*

- *InstallScript Object*

The AddSetupType method creates a setup type with the specified name in the current project.

Syntax

```
AddSetupType (Name As String) As ISWiSetupType
```

Parameters

Table 12-11 • AddSetupType Method Parameters

Parameter	Description
Name	Pass the name for the setup type that you want to add. The name should be the text that you want to use as the display name. The automation interface automatically generates a valid primary key for the new setup type based on the name that you specify for this parameter.

Example

The following Visual Basic lines demonstrate the AddSetupType method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pSetupType As ISWiSetupType

Set pSetupType = pProj.AddSetupType ("Client Setup - Save Space")

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

AddSQLServerConnection Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the AddSQLServerConnection method to add a new server connection to your project.

Syntax

```
Function AddSQLServerConnection(strConnectionName As String) As ISWiSQLConnection
```

Parameters

Table 12-12 • AddSQLServerConnection Method Parameters

Parameter	Description
ConnectionName	Specify the name of the new connection you want to establish.

Return Values

This method returns ISWiSQLConnection object.

Applies To

- ISWiProject

AddUpgradeTableEntry Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the AddUpgradeTableEntry method to create a new ISWiUpgradeTableEntry item and return a handle to that object.

Syntax

```
AddUpgradeTableEntry()
```

Example

The following Visual Basic lines demonstrate this method:

```
'################################################
'# THIS SAMPLE WILL NOT EXECUTE PROPERLY WITHOUT A VALID PROJECT PATH
'################################################
set pProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProject.OpenProject("C:\Documents and Settings\Testlab\My Documents\MySetups\your project name-
21.ism")


'################################################
'To create a new entry

dim pTableEntry
set pTableEntry = pProject.AddUpgradeTableEntry()

pTableEntry.UpgradeCode ="{12345678-1234-1234-1234-123456789012}"
```

```
pTableEntry.VersionMin = "1.0"
pTableEntry.VersionMax = "2.0"
pTableEntry.Language = "0"
pTableEntry.Remove = ""
pTableEntry.ActionProperty = "ACTIONPROP_AUTOMATION"
pTableEntry.Attributes = 5

'###############################################
'To find and delete an existing entry

set pTableEntry =pProject.IswiUpgradeTableEntries("{12345678-1234-1234-1234-123456789012}")
pTableEntry.Delete

pProject.SaveProject
pProject.CloseProject
```

Applies To

- ISWiProject

BuildPatchConfiguration Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call BuildPatchConfiguration to build the patch configuration identified by the strPatchConfiguration parameter. This action has the same effect as right-clicking on a patch configuration in the Patch Design view and then selecting Build a Patch.

Syntax

```
BuildPatchConfiguration (strPatchConfigName as string)
```

Parameters

Table 12-13 • BuildPatchConfiguration Method Parameters

Parameter	Description
strPatchConfigName	Enter the fully qualified path to your project.

Properties

Table 12-14 • Properties Associated with BuildPatchConfiguration Method

Property	Description
PatchConfigErrorCount	Include the error count associated with a particular build.
PatchConfigWarningCount	Include the warning count associated with a particular build.

Example

The following Visual Basic lines demonstrate this method:

```
'Create the project object
dim pProject
set pProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")


'Call the OpenProject method
'########################################################################
'# THIS SAMPLE WILL NOT EXECUTE PROPERLY WITHOUT A VALID PROJECT PATH
'########################################################################
'pProject.OpenProject("\Your project.ism")

pProject.OpenProject("C:\Documents and Settings\Testlab\My Documents\MySetups\your project name-
21.ism")

'Call the OpenProject method
'###############################################################################
'# THIS SAMPLE WILL NOT EXECUTE PROPERLY WITHOUT A VALID PATCH CONFIGURATION NAME
'###############################################################################

pProject.BuildPatchConfiguration "Config1"

MsgBox "Patch Congifuration Config1 build with " & _
     pProject.PatchConfigErrorCount & " Errors and " & _
     pProject.PatchConfigWarningCount & " Warnings"
```

Applies To

- ISWiProject

BuildPCPFile Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The BuildPCPFile method builds a patch package (.msp) file based on the settings in a patch creation properties (.pcp) file.

You do not need to have a project open in order to create a patch.

Syntax

```
BuildPCPFile (strPCPFile As String, strPatchFile As String, strLogFile As String, bCreateUpdateExe As Boolean) As Long
```

Parameters

Table 12-15 • BuildPCPFile Method Parameters

Parameter	Description
strPCPFile	Specify the fully qualified path to a .pcp file. You can create or modify a .pcp file through the Patch Design view.
strPatchFile	Specify the fully qualified path for the .msp file you want to create. Any existing file with this name is overwritten. The file cannot be created if the directory you provided does not exist.
strLogFile	Specify the fully qualified path to the .log file that records this method's status and success in creating the patch package. If the file exists, the current status is appended to it.
bCreateUpdateExe	Set this option to True to have the package wrapped in an executable that applies the patch.

Return Values

BuildPCPFile() returns the result returned by the Windows Installer function UiCreatePatchPackage.

The BuildPCPFile method builds a patch package (.msp) file based on the settings in a patch creation properties (.pcp) file.

You do not need to have a project open in order to create a patch.

Applies To

- ISWiProject

CloseProject Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

To close the project file (.ism or .dim) that you opened with OpenProject, call CloseProject.

Syntax

```
CloseProject ()
```

Return Values

CloseProject always returns 0.

Applies To

- ISWiProject

CreatePatch Method



Caution • *This method is deprecated for new development and has been replaced with the BuildPCPFile and BuildPatchConfiguration methods. To facilitate backward compatibility with existing build scripts, the CreatePatch method calls the BuildPCPFile method.*



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Syntax

```
CreatePatch()
```

Applies To

- ISWiProject

CreateProject Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Project-specific differences are noted where appropriate.

The CreateProject method creates a new InstallShield project file (.ism) for a Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, or Merge Module project; it create a new InstallShield DIM project file (.dim) for DIM projects.

Syntax

```
CreateProject (strISWiProjectFile As String, ByVal projectType As ISWiProjectType)
```

Parameters

Table 12-16 • CreateProject Method Parameters

Parameter	Description
strISWiProjectFile	Enter the fully qualified path to the .ism or .dim file that you want to create. Include the full file name.
projectType	Specify what type of project you want to create. Valid values are: • **eptMsi (1)**—Creates a Basic MSI project. • **eptScript (–1)**—Creates an InstallScript MSI project. • **eptPro (9)**—Creates an InstallScript project. • **eptProObj (10)**—Creates an InstallScript Object project. • **eptMsm (2)**—Creates a Merge Module project. • **eptDim (15)**—Creates a DIM project.

Example

The following sample code creates a Basic MSI project called **TestProject.ism**.

```
Dim pProject As ISWiAutoAutomation Interface Version.ISWiProject
Set pProject = CreateObject("ISWiAutoAutomation Interface Version.ISWiProject")

Dim sProjectName As String
sProjectName = "C:\InstallShield 2016 Projects\TestProject.ism"

pProject.CreateProject sProjectName, eptMsi
```

```
pProject.OpenProject sProjectName, False

pProject.CloseProject
```

Applies To

- ISWiProject

DeleteAdvancedFile Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The DeleteAdvancedFile method deletes the specified advanced file from the current project.

Syntax

```
DeleteAdvancedFile (pAdvancedFile As ISWiAdvancedFile) As Long
```

Parameters

Table 12-17 • DeleteAdvancedFile Method Parameters

Parameter	Description
pAdvancedFile	Specify the advanced file object to be deleted.

Example

The following Visual Basic lines demonstrate the DeleteAdvancedFile method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pAdvancedFile As ISWiAdvancedFile

Set pAdvancedFile = pProj.ISWiAdvancedFiles.Item(1)
pProj.DeleteAdvancedFile pAdvancedFile

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

DeleteComponent Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call DeleteComponent to delete a component in the project. Similar to deleting a component in the Setup Design views, DeleteComponent both removes a component from its feature and permanently deletes it from the project.

Syntax

```
DeleteComponent (Component As ISWiComponent)
```

Parameters

Table 12-18 • DeleteComponent Method Parameters

Parameter	Description
Component	Specify the ISWiComponent object that the method will delete.

Example

You can use the following lines to delete all components that are associated with the feature FeatureName1:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"

Dim pFeat As ISWiFeature
Dim pComp As ISWiComponent

Set pFeat = pProj.ISWiFeatures.Item("FeatureName1")

For Each pComp In pFeat.ISWiComponents
    pProj.DeleteComponent pComp
Next

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

DeleteCustomAction Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Call the DeleteCustomAction method to completely delete a custom action from the current project.

Syntax

```
DeleteCustomAction (CustomAction As ISWiCustomAction)
```

Parameters

Table 12-19 • DeleteCustomAction Method Parameters

Parameter	Description
CustomAction	Pass the ISWiCustomAction object you want to delete from the current project.

Applies To

- ISWiProject

DeleteMergeModule Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the DeleteMergeModule method to completely delete a merge module from the current project.

Syntax

```
DeleteMergeModule (FileName As String)
```

Parameters

Table 12-20 • DeleteMergeModule Method Parameters

Parameter	Description
FileName	Pass the fully qualified file name for the merge module (.msm) file that you want to delete from the project.

Applies To

- ISWiProject Object

DeletePathVariable Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the DeletePathVariable method to delete a path variable from your project. Using this method is similar to deleting a path variable from the Path Variables view in InstallShield.



Important • *Predefined path variables cannot be modified or deleted. If you attempt to do so, exception error 3142 occurs. For more information, see Predefined Path Variables.*

Syntax

```
DeletePathVariable (pPathVar ISWiPathVariable) As Long
```

Parameters

Table 12-21 • DeletePathVariable Method Parameters

Parameter	Description
pPathVar	Pass the ISWiPathVariable object for the path variable that you want to delete.

Applies To

- ISWiProject

DeleteProductConfig Method

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

Call the DeleteProductConfig method to delete a product configuration from your project. Using this method is similar to deleting a product configuration in the Releases view of the IDE.

Syntax

```
DeleteProductConfig (ProductConfig As ISWiProductConfig)
```

Parameters

Table 12-22 • DeleteProductConfig Method Parameters

Parameter	Description
ProductConfigKey	Pass the name of the product configuration that you want to delete.

Applies To

- ISWiProject

DeleteProperty Method

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Call the DeleteProperty method to delete a property from the Property table. The function returns 0 if successful.

Syntax

```
Public Function DeleteProperty(ByVal pProperty As ISWiProperty) As Long
```

Parameters

Table 12-23 • DeleteProperty Method Parameters

Parameter	Description
Property	Pass the name of the property you want to delete.

Example

```
Dim proj As IswiAutoAutomation Interface Version.ISWiProject
Set proj = New IswiAutoAutomation Interface Version.ISWiProject
proj.OpenProject "c:\mysetups\your project name-3.ism"
Dim pProp As IswiAutoAutomation Interface Version.ISWiProperty
Set pProp = proj.ISWiProperties.Item("MyProperty")
proj.DeleteProperty pProp
```

Applies To

- ISWiProject

DeleteSetupFile Method



Project • *The DeleteSetupFile method applies to the following project types.*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

The DeleteSetupFile method deletes the specified support file from the current project.

Syntax

```
DeleteSetupFile (pSetupFile As ISWiSetupFile) As Long
```

Parameters

Table 12-24 • DeleteSetupFile Method Parameters

Parameter	Description
pSetupFile	The support file object to be deleted.

Example

The following Visual Basic lines demonstrate the DeleteSetupFile method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pSetupFile As ISWiSetupFile
```

```
Set pSetupFile = pProj.ISWiSetupFiles.Item("MySupportFile.abc")
pProj.DeleteSetupFile pSetupFile

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

DeleteSetupType Method



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The DeleteSetupType method deletes the specified setup type from the current project.

Syntax

```
DeleteSetupType (pSetupType As ISWiSetupType) As Long
```

Parameters

Table 12-25 • DeleteSetupType Method Parameters

Parameter	Description
pSetupType	The setup type object to be deleted.

Example

The following Visual Basic lines demonstrate the DeleteSetupType method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pSetupType As ISWiSetupType

Set pSetupType = pProj.ISWiSetupTypes.Item(1)
pProj.DeleteSetupType pSetupType

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

DeleteSQLConnection Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the DeleteSQLConnection method to remove a SQL Server Connection from the project.

Syntax

```
Function DeleteSQLConnection(Connection As ISWiSQLConnection) As Long
```

Parameters

Table 12-26 • DeleteSQLConnection Method Parameters

Parameter	Description
Connection	Specify the connection you want to remove from the project.

Applies To

- ISWiProject

DeselectLanguage Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the DeselectLanguage method to deselect a language from the current project.

Syntax

```
DeselectLanguage (LangId As String)
```

Parameters

Table 12-27 • DeselectLanguage Method Parameters

Parameter	Description
LangId	Pass the language ID you want to deselect in a string, as in oProject.DeleteLanguage "1036".

Applies To

- ISWiProject

ExportProject Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call ExportProject to convert an .ism file to XML format before checking it into a source code control program.



Note • *When a project file format is converted to either XML or binary format, the project file extension remains *.ism.*

The .isv file format has been deprecated to version 2.x of InstallShield for Windows Installer and versions of InstallShield Developer prior to 8.x.

Syntax

```
ExportProject (strISWiTextProjectFile As String, Optional strISWiProjectFile As String) As Long
```

Parameters

Table 12-28 • ExportProject Method Parameters

Parameter	Description
strISWiTextProjectFile	Enter the fully qualified path to the .isv file.
strISWiProjectFile	Enter the fully qualified path to the .ism file. This parameter is optional only if the .ism file is already open.

Applies To

- ISWiProject

ExportStrings Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call ExportStrings to copy all or a portion of the string entries for a specific language to a tab-delimited Unicode text file (.txt). You can provide the .txt file to a translator who can update the file with translated text. After it has been translated, you can import the translated string entries into your project by calling the ImportStrings method.

Syntax

```
ExportStrings (strfile As String, dTimeStamp As Date, sLang As String) As Boolean
```

Parameters

Table 12-29 • `ExportStrings Method Parameters`

Parameter	Description
strfile	Enter the fully qualified path to the text file that will contain the exported strings. If the file exists, the automation interface overwrites it. ExportStrings creates a Unicode file.
dTimeStamp	All string entries have a date and time that they were last modified. This timestamp is useful if you want to give the translator only the string entries that are new or have been modified since the last round of translations. To export only the string entries that have been modified since a specific date, specify the earliest date for the string entries you want to export. The automation interface exports the string entries that have changed on or since that date. To export all of the entries for the specified language, enter the number 0.
sLang	Specify the identifier for the language whose string entries you want to export to the text file.

Applies To

- ISWiProject

ForceUpgrade Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The ForceUpgrade method upgrades the project from an earlier version of InstallShield to the current version.

Syntax

```
ForceUpgrade (ISWiProjectFile As String)
```

Parameters

Table 12-30 • ForceUpgrade Method Parameters

Parameter	Description
ISWiProjectFile	Enter the fully qualified path to the project file (.ism) that you want to upgrade. Include the full file name.

Applies To

- ISWiProject

GenerateGUID Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The GenerateGUID method generates a new, unique GUID that is returned as a string.

Applies To

- ISWiProject

ImportProject Method



Note • *This method has been deprecated to version 2.x of InstallShield for Windows Installer and versions of InstallShield Developer prior to 8.x.*

Call ImportProject to convert an .isv file to an .ism file. If you access your project from a source code control program, InstallShield saves it as a text (.isv) file. However, OpenProject and ISCmdBld.exe both require a standard InstallShield project (.ism) file.

Later, you can call ExportProject to convert the .ism file back to an .isv file before checking it in to your source control program.

ImportProject fails if the project is already open in the automation interface or in InstallShield.

Syntax

```
ImportProject (strISWiProjectFile As String, strISWiTextProjectFile As String) As Long
```

Parameters

Table 12-31 • ImportProject Method Parameters

Parameter	Description
strISWiProjectFile	Enter the fully qualified path to the resulting .ism file.
strISWiTextProjectFile	Enter the fully qualified path to the .isv file.

Error Codes

Table 12-32 • ImportProject Method Error Codes

Code	Description
1007	The project file is already open.
1009	Could not import the project.
1012	The .isv file is missing.

Applies To

- ISWiProject

ImportStrings Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call ImportStrings to import the contents of a tab-delimited text file (.txt) into the project for a specific language. This method is useful when you have exported the entries to a text file and had them translated.

Syntax

```
ImportStrings (strfile As String, sLang As Integer, Optional Enum eiType, Optional strLogFile As String)
As Boolean
```

Parameters

Table 12-33 • ImportStrings Method Parameters

Parameter	Description
strfile	Enter the fully qualified path to the text file that contains the string entries that you want to import.
sLang	Specify the identifier for the language whose string entries you want to import to the project.
eiType	Specify how you want to handle conflicts when importing string entries: • **eiIgnore (0)**—Do not replace the string entry's value if it already exists in the project for the specified language. • **eiOverwrite (1)**—Replace the project's existing string entry with the one in the text file. This is the default value for this optional parameter.
strLogFile	Specify the fully qualified path to the log file that should contain the status and success of importing the string entries. The log file records any conflicts in importing string entries and lists any improperly formatted, hence unusable, entries. If the log file exists, it is overwritten.  ***Note*** • *This parameter is optional.*

Applies To

- ISWiProject

OpenProject Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

To open your project so that you can modify it through the automation interface, call OpenProject with the fully qualified path to your .ism or .dim file. This method is equivalent to loading a project in InstallShield.



Note • *The ISWiProject.OpenProject method fails if the project is open in InstallShield.*

Syntax

```
OpenProject (strISWiProjectFile As String, Optional ByVal bReadOnly As Boolean) As Long
```

Parameters

Table 12-34 • OpenProject Method Parameters

Parameter	Description
strISWiProjectFile	Enter the fully qualified path to the project file (.ism or .dim).
bReadOnly	Determines whether the project is opened with write privileges. True opens the project as a read-only file. The default value for this optional parameter is False.

Return Values and Errors

Table 12-35 • OpenProject Method Return Values and Errors

Return Value	Description
0	Project successfully opened.
1	Project opened, but is locked by some other process.
2	Attempted to open project for write access, but project opened in read-only mode.
3	There was a problem creating the merge module catalog. Verify that the merge module search path is correct (via the File Locations tab of the Options dialog box or using -o for the Standalone Build), verify that no merge modules on the merge module path are corrupt, or self-register **IsMMUpdater2.dll**.
Error Code 1100	Unable to open file: %1. Ensure that the file is a valid InstallShield project.
Error Code 1101	Unable to open file: %1. This project was created using a previous version of InstallShield Developer, InstallShield–Windows Installer or InstallShield Express.
Error Code 1102	Unable to open file: %1. This project was created using InstallShield Professional or InstallShield Express 2.xx.
Error Code 1103	Unable to open file: %1. The project was created using a more recent version of InstallShield Developer or the project has been updated to a more recent version of InstallShield, InstallShield DevStudio, or InstallShield Developer.

Applies To

- ISWiProject

RemoveFeature Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The RemoveFeature method permanently deletes a feature from the current project.

Syntax

```
RemoveFeature (Feature As ISWiFeature) As Long
```

Parameters

Table 12-36 • RemoveFeature Method Parameters

Parameter	Description
Feature	The item in ISWiFeatures that you want to remove.

Example

The following code fragment opens a project, deletes the feature named NewFeature1, and then saves and closes the project:

```
Dim pProj As ISWiProject
Dim pFeat As ISWiFeature

Set pProj = New ISWiProject
pProj.OpenProject "C:\Test\SampleProject.ism"

Set pFeat = pProj.ISWiFeatures.Item("NewFeature1")
pProj.RemoveFeature pFeat

pProj.SaveProject
pProj.CloseProject
```

When RemoveFeature is a member of ISWiFeature, it cannot be used to delete the current feature. However, you can call it to delete a subfeature, such as the first subfeature under NewFeature1 in the following example:

```
Set pFeat = pProj.ISWiFeatures.Item("NewFeature1")
pProj.RemoveFeature pFeat.Features(1)
```

Applies To

- ISWiProject
- ISWiFeature

SaveProject Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

To save any changes that you have made to your project file through the automation interface, call the SaveProject method.

Syntax

```
SaveProject ()
```

Return Values and Errors

Table 12-37 • SaveProject Method Values

Return Value	Description
ERROR_SUCCESS (0)	Project is saved.
Error Code 1104	Unable to save file: %1. The project is open in read-only mode or is being used by another process.

Applies To

- ISWiProject

ISWiAdvancedFile Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

ISWiAdvancedFile represents an advanced file in your project. The advanced file can be an existing one from the Support Files view (or the Support Files/Billboards view) in InstallShield or one added by calling AddAdvancedFile.

Use the project's ISWiAdvancedFiles collection to access a specific advanced file. You must specify either an index number or the file key for the Item property of ISWiAdvancedFiles.

Members

Table 12-38 • ISWiAdvancedFile Object Members

Name	Type	Description
Name	Read-Only Property	Stores the file key of the advanced file. See the description of ISWiFile above for more information about the file key.
BuildSourcePath	Read-Write Property	Stores the advanced file's fully qualified source filename. Any path variables are resolved to the actual path.
Disk	Read-Write Property	This enumerated integer property specifies the Advanced Files folder in which you want to add the advanced file. Specify one of the following values: • **edtDisk1 (1)**—Specifies the Disk 1 folder. • **edtLastDisk (-1)**—Specifies the Last Disk folder. • **edtOther (0)**—Specifies the Other folder.

Example

The file key is visible in the IDE only as the first entry in the Direct Editor view's ISDisk1 table. Instead of using the file key, you could write code similar to the following example to check for a certain file name:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pAdvancedFile As ISWiAdvancedFile

For Each pAdvancedFile In pProj.ISWiAdvancedFiles
    If UCase(pAdvancedFile.BuildSourcePath) = "C:\MY FILES\MYADVANCEDFILE.EXT" Then
        Exit For
    End If
Next

pProj.SaveProject
pProj.CloseProject
```

If the file **C:\My Files\MyAdvancedFile**.ext is found among the project's setup files, the For loop ends, and pAdvancedFile holds a copy of the ISWiAdvancedFile object for **C:\My Files\MyAdvancedFile.ext**.

Applies To

- ISWiProject

ISWiAutomaticUpgradeEntry Object



Project • *This information applies to the following project types:*

- *Basic MSI*

- *InstallScript MSI*

The ISWiAutomaticUpgradeEntry object represents an automatic upgrade item in the Upgrades view of the InstallShield interface.

Members

Table 12-39 • ISWiAutomaticUpgradeEntry Object Members

Name	Type	Description
Delete	Method	Deletes the AutomaticUpgradeEntry from the project file.
Name	Read-Write Property	Sets the name of the upgrade item as displayed in the IDE.
UpgradedSetupPath	Read-Write Property	Sets the path to the MSI package that needs to be upgraded. The build engine will create the settings required to upgrade this setup.
ReleaseFlag	Read-Write Property	Configure this upgrade entry to only be built into the setup when a certain release flag has been set.

Applies To

- ISWiProject

Delete Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call this method to delete the ISWiAutomaticUpgradeEntry from the project file.

Syntax

```
Delete()
```

Applies To

- ISWiAutomaticUpgradeEntry

ISWiCustomAction Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

ISWiCustomAction represents a custom action in the Custom Actions and Sequences view (or the Custom Actions view) of InstallShield. You can retrieve a custom action by specifying an item in the ISWiCustomActions collection.

Members

Table 12-40 • ISWiCustomAction Object Members

Property	Type	Description
ActionType	Read-Write Property	Holds the numeric custom action type. For more information, see Custom Action Type Overview.
Comment	Read-Write Property	An optional comment for the custom action.
Name	Read-Write Property	The name of the custom action.
Source	Read-Write Property	Specifies the source of the custom action. See Creating Custom Actions in the Custom Actions and Sequences View (or the Custom Actions View) for the required Source property for different types of custom actions.
SourceEx	Write-Only Property	Sets a custom action source property, same as the Source property. SourceEx replaces the existing binary record for the new path when you change a source that is stored in the Binary table. If other custom actions use the path, their path is also updated. Source creates a new binary record when other custom actions use the path. Only the path in the custom action is updated.
Target	Read-Write Property	Specifies the target of the custom action. See Creating Custom Actions in the Custom Actions and Sequences View (or the Custom Actions View) for the required Target property for different types of custom actions.

Applies To

- ISWiProject

ISWiComponent Object



Project • *This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Project-specific differences are noted where appropriate.

ISWiComponent represents a component in the InstallShield user interface. You can retrieve a component by specifying an item in the ISWiComponents collection. The component can be one created in the IDE or as a result of calling AddComponent.

Many component attributes are available in the automation interface through this object's methods and properties. Refer to the following table to see which methods, properties, and collections are available for your project type.

Members

Table 12-41 • ISWiComponent Object Members

Name	Project	Type	Description
AddComponentSubFolder	InstallScript, InstallScript Object	Method	Adds a subfolder to the current ISWiComponent.
AddDynamicFileLinking	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Method	Adds a dynamic file link to the current component.
AddEnvironmentVar	Basic MSI, InstallScript MSI, Merge Module	Method	Adds an environment variable to the current component.
AddFile	All	Method	Adds a file to the current component.
AddRemoveFile	Basic MSI, InstallScript MSI, Merge Module	Method	Adds a file or folder removal entry to the current component.
Attrib64BitComponent	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Set this property to True to have the component registered as 64-bit. Setting this property to False causes the component to be registered as 32-bit. If the component is 64-bit and is replacing a 32-bit component, be sure to provide a new GUID via the GUID property. Note that 64-bit support requires version 2.0 of the Windows Installer service.
Comments	All	Read-Write Property	Gets or sets the comments for this component.
Compressed	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets the component's Compressed property, which specifies whether to compress the component's files (True) or leave them uncompressed (False) when the files are stored in a cabinet file by the release builder.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
Condition	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Contains any condition associated with the component.
Destination	All	Read-Write Property	Gets or sets the component's destination. The following formats are valid: • (Windows Installer based projects only:) [INSTALLDIR] • (Windows Installer based projects only:) [ProgramFilesFolder]Subfolder • (InstallScript projects only:) <TARGETDIR> • (InstallScript projects only:) <WINDIR>\Subfolder • C:\FolderName  ***Important*** • *Setting the Destination property for the ISWiComponent object resets the object's SourceLocation property. Therefore, any code that sets the SourceLocation property should be placed after any code that sets the Destination property.*
DeviceDriverFlags	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property is no longer supported.
DeviceDriverFlagsEx	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property enables you to specify the flag settings for device drivers and run-time options directly from the automation layer.
DeviceDriverSetRedist	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property sets the .dll used at run time for all device driver installations.
DeviceDriverUse64BitRedist	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property is no longer supported.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
DeviceDriverUseLocalizedRedist	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property is no longer supported.
Difference	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets the component's Difference property, which specifies whether to accept (True) or override (False) the default behavior of the media builder when building a differential media—which is to exclude a file in this component from the differential media if the same file (with the same date and time, size, and attributes) exists in each of the specified comparison media.
DotNetApplicationFile	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	To install your assembly as private, set this attribute to the FileKey of the file that contains your assembly. If this property is set to NULL, your assembly will be installed to the Global Assembly Cache.
DotNetAssembly	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets the component's .NET Assembly property, which specifies whether the component's files are installed as local .NET assemblies (True) or not installed as assemblies (False).
DotNetCOMInterop	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Set this to True if your .NET assembly needs to be available from COM calls.
DotNetInstallerClass	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Set this to True if your assembly implements methods from the .NET Installer namespace.
DotNetInstallerClassArgCommit	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property allows you to specify argument lists for the Installer class when the Commit method is called.
DotNetInstallerClassArgInstall	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property allows you to specify the argument lists for the Install execution context for the Installer class.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
DotNetInstallerClassArgRollback	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property allows you to specify argument lists for the Installer class when the Rollback method is called.
DotNetInstallerClassArgUninstall	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property allows you to specify the argument lists for the Uninstall execution context for the Installer class.
DotNetPrecompile	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	If this property is set to True, at installation the setup creates a native image from the .NET assembly and installs it to the native image cache on the target system. This allows the assembly to load and execute faster, because it restores code and data structures from the native image cache rather than from just-in-time (JIT) compilation.
DotNetScanAtBuild	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	This property instructs the build engine on how to handle your assembly. Set it to one of the following values: • **ednbNone (0)**—Build does nothing. • **ednbProps (1)**—Build scans your assembly for properties. • **ednbDepAndProps (2)**—Build scans your assembly for properties and dependencies.
ExtractAtBuild	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the component's COM Extract at Build setting. Possible values are True and False.
FTPLocation	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the component's FTP Location property, which specifies an FTP location for the component.
GUID	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Holds a string GUID, also known as the Component Code property, surrounded by braces—{12345678-1234-1234-1234-1234567890AB}, for example.
HTTPLocation	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the component's HTTP Location property, which specifies an HTTP location for the component.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
ImportINIFile	All	Method	Imports INI files from the automation layer.
ImportRegFile	All	Method	Merges the contents of a REG file with the registry data for this component.
IsDeviceDriver	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	If this Boolean property is true, the key file for this component is a device driver package; if the property is false, the key file is not a device driver package.
IsPlatformSelected	InstallScript, InstallScript Object	Read-Write Property	In conjunction with the PlatformSuiteCheck property, gets or sets the component's Platform Suite(s) property, which specifies to what platform suites, if any, the component is specific. Use the following constants to set this property: • epsMSBackOffice = 32 (&H20) • epsMSSmallBusinessServer = 256 (&H100) • epsMSSmallBusinessServerWithRestrictiveLiscenses = 512 (&H200) • epsTerminalServices = 16 (&H10) • epsWin2kAdvSvrOrWinDotNetEnterpriseServer = 128 (&H80) • epsWin2kOrWinDotNetDatacenterServer = 64 (&H40) • epsWinServer = 4 • epsWinWorkstation = 8 • epsWinXPHome = 2 • epsWinXPPro = 1 • epsArchIa64 = 1024 (&H400) • epsArchAmd64 = 2048 (&H800) • epsArchIntel32 = 4096 (&H1000) • epsWinServer2003R2 = 8192 (&H2000) For example: `pFeature.IsPlatformSelected(epsWinXPHome) = True`

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
ISWiComponentSubFolders	InstallScript, InstallScript Object	Collection	Contains all of the subfolders associated with this component.
ISWiDynamicFileLinkings	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	Collection	Contains all of the dynamic file links that are associated with a specified component.
ISWiEnvironmentVars	Basic MSI, DIM, InstallScript MSI, Merge Module	Collection	Contains all of the environment variables that are associated with this component.
ISWiFiles	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Collection	Contains all of the files associated with this component.
ISWiFolders	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Collection	Contains all of the folders associated with this component.
ISWiRemoveFiles	Basic MSI, DIM, InstallScript MSI, Merge Module	Method	Returns an ISWiRemoveFiles collection that contains all of the file and folder removal entries that are associated with the current component.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description

KeyPath	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	
			Note • *ISWiComponent.KeyPathType must be set before setting ISWiComponent.KeyPath. Errors result if this is done in the reverse order.*
			Contains the file or registry entry that serves as this component's key path. The exact type is indicated by the KeyPathType property, below.
			When the key path is a key file, it contains the file key. You can view this value in the Key column of a component's file list, but be aware that file keys are subject to change when the files are dynamically linked. The file key is the same as the Name property of an ISWiFile object.
			When the key path is a registry value, there is a specially formatted string that contains the registry key. The path to the registry key is enclosed in square brackets and separated from the value name with a vertical bar (or pipe character). For example, the following lines set the key path for the component m_MyComponent to the value MyName under HKEY_LOCAL_MACHINE\Software\MyCompany\Settings:
			`m_MyComponent.KeyPathType = kptRegistry`
			`m_MyComponent.KeyPath = "[HKEY_LOCAL_MACHINE\Software\MyCompany\Settings]|MyName"`
			A file cannot serve as the key file if it is dynamically linked.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
KeyPathType	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	ISWiComponent.KeyPathType *must* be set before setting ISWiComponent.KeyPath. Errors result if this is done in the reverse order.Stores a value that points to the precise type of key path contained in the KeyPath property. Specify one of the following types: • kptRegistry (1) • kptFile (2) • kptFolder (3) • kptODBC (4) Note that the automation interface can set the key path type only to kptRegistry or kptFile, but the other values are possible return values if you are reading the current key path type.
Miscellaneous	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the component's Miscellaneous property, which associates a text string with the component.
Name	All	Read-Only Property	Component name of the current item in the collection.
NeverOverwrite	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Instructs the installer whether to replace an existing component. Specify True to leave the existing component or False to replace it.
OSFilter	InstallScript, InstallScript MSI, InstallScript Object	Read-Write Property	Gets or sets the component's Operating Systems setting. If the target machine's operating system does not match one of the operating systems that are specified for this setting, the component is not installed. By default, components are operating system independent, meaning that none of the component's data are specific to certain operating systems.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
OSFilter (cont.)			Use the following constants to set this property: • eosOSIndepedent = 0 • eosWin95 = &H10 (16) • eosWin98 = &H40 (64) • eosWinMe = &H80 (128) • eosWinNT4 = &H10000 (65536) • eosWin2000 = &H100000 (1048576) • eosWinXP = &H400000 (4194304) • eosWinServer2003 = &H800000 (8388608) • eosWinVista = &H1000000 (16777216)—These constants are for Windows Vista and Windows Server 2008. • eosWin7 = &H2000000 (33554432)—These constants are for Windows 7 and Windows Server 2008 R2. • eosWin8 = &H4000000 (67108864)—These constants are for Windows 8 and Windows Server 2012. • eosWin81 = &H8000000 (134217728)—These constants are for Windows 8.1 and Windows Server 2012 R2. • eosAll = &HFD100D0 (265355472) You can specify multiple platforms; for example, **eosWin7 Or eosWinServer2008 Or eosWinVista Or eosWinServer2003**.  ***Tip*** • *Use this property with the IsPlatformSelected property to distinguish between Windows Vista and Windows Server 2008, or between Windows 7 and Windows Server 2008 R2, or between Windows 8 and Windows Server 2012, or between Windows 8.1 and Windows Server 2012 R2.*

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
OverwriteMainOptions	InstallScript, InstallScript Object	Read-Write Property	In conjunction with the OverwriteSubOptionsVersion and OverwriteSubOptionsDate properties, gets or sets the component's Overwrite property, which specifies whether the component's files overwrite already-existing versions on the target system always, never, or conditionally based on date/time stamp or version number. Use the following constants to set this property: • **ecomoAlways (0)**—Files on the target system are always overwritten. • **ecomoByDate (1)**—Files on the target system are conditionally overwritten based on date/time stamp as specified by the OverwriteSubOptionsDate property. • **ecomoByVersion (2)**—Files on the target system are conditionally overwritten based on version number as specified by the OverwriteSubOptionsVersion property. • **ecomoByVersionThenDate (3)**—Files on the target system are conditionally overwritten based on version number or, if the file on your distribution media and the file on the target system have the same version number or if neither file has a version number, by date/time stamp as specified by the OverwriteSubOptionsVersion and OverwriteSubOptionsDate properties. • **ecomoNever (4)**—Files on the target system are never overwritten.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
OverwriteSubOptionsDate	InstallScript, InstallScript Object	Read-Write Property	In conjunction with the OverwriteMainOptions and OverwriteSubOptionsVersion properties, gets or sets the component's Overwrite property, which specifies whether the component's files overwrite already-existing versions on the target system always, never, or conditionally based on date/time stamp or version number. Ignored if OverwriteMainOptions is set to ecomoAlways, ecomoByVersion, or ecomoNever. Use the following constants to set this property: • **ecomsoNewer (0)**—A file on the target system is overwritten if the file on your distribution media has a more recent date and time stamp. • **ecomsoNewerOrSame (1)**—A file on the target system is overwritten if the file on your distribution media has a more recent or the same date and time stamp. • **ecomsoOlder (2)**—A file on the target system is overwritten if the file on your distribution media has a less recent date and time stamp.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
OverwriteSubOptionsVersion	InstallScript, InstallScript Object	Read-Write Property	In conjunction with the OverwriteMainOptions and OverwriteSubOptionsDate properties, gets or sets the component's Overwrite property, which specifies whether the component's files overwrite already-existing versions on the target system always, never, or conditionally based on date/time stamp or version number. Ignored if OverwriteMainOptions is set to ecomoAlways, ecomoByDate, or ecomoNever. Use the following constants to set this property: • **ecomsoNewer (0)**—A file on the target system is overwritten if the file on your distribution media has a higher version number. • **ecomsoNewerOrSame (1)**—A file on the target system is overwritten if the file on your distribution media has a higher or the same version number. • **ecomsoOlder (2)**—A file on the target system is overwritten if the file on your distribution media has a lower version number.
Permanent	All	Read-Write Property	This property corresponds to the Permanent component property in the IDE. Set Permanent to True to make sure the component is never removed; False otherwise.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
PlatformSuiteCheck	InstallScript, InstallScript Object	Read-Write Property	In conjunction with the IsPlatformSelected property, gets or sets the component's Platform Suite(s) property, which specifies to what platform suites, if any, the component is specific. Use the following constants to set this property: • **ecpscAll (0)**—The setup installs the component's files only if all of the suites that are specified by the IsPlatformSelected property exist on the target system. • **ecpscAtLeastOne (1)**—The setup installs the component's files only if at least one of the suites that are specified by the IsPlatformSelected property exist on the target system. • **ecpscSuiteIndependent (2)**—Installation of the component's files does not depend on the target system's suite.
PotentiallyLocked	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets the component's Potentially Locked property, which specifies whether already-installed versions of the component's files may be locked—that is, in use by another application—during setup.
RegFileToMergeAtBuild	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Provides the complete path to a .reg file to have it merged with the component's registry entries at build time.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
RegistrationType	Basic MSI, InstallScript MSI, Merge Module	Property	The following options are available for determining the source of the registration information for this component: • **rgtNone (0)**—InstallShield will not dynamically extract the data or mark the component as self-registering, but any information in the COM Registration advanced setting will be registered. • **rgtSelfReg (-1)**—The files in this component will have their self-registration routines invoked by the installer. • **rgtExtractAtBuild (-2)**—Dynamically extracts COM registration information when you build a release. Because the Registration Type property has been removed from the component views in the IDE, setting the RegistrationType property in the automation interface has the following effects: • Using rgtNone sets the component's COM Extract at Build property to No. • Using rgtSelfReg sets the component's COM Extract at Build property to No, and sets each file in the component to use self-registration. • Using rgtExtractAtBuild sets the component's COM Extract at Build property to Yes, and sets each file in the component not to use self-registration.
RemoteInstallation	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Use one of the following options to install the component's files locally or leave them on the distribution medium: • **rfsLocal (0)**—Installs the files to the folder stored in the component's Destination property. • **rfsSource (1)**—Leaves the files on the distribution medium. • **rfsOptional (2)**—The component follows the feature's property.

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
RemoveComponentSubFolder	InstallScript, InstallScript Object	Method	Removes a subfolder to the current ISWiComponent.
RemoveDynamicFileLinking	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Method	Removes a dynamic file link from the current component.
RemoveEnvironmentVar	Basic MSI, InstallScript MSI, Merge Module	Method	Removes an environment variable from the current component.
RemoveFile	All	Method	Removes a file from the current component.
RemoveRemoveFile	Basic MSI, DIM, InstallScript MSI, Merge Module	Method	Removes a file or folder removal entry from the current component. To remove a file or folder removal entry from the project, pass the ISWiRemoveFile object.
SelfRegister	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets the component's Self-Register property, which specifies whether the component's files are self-registering.
SharedDLLRefCount	All	Read-Write Property	True indicates that the installer should increment a reference count for each file in this component when it is installed, False otherwise.
SourceLocation	All	Read-Write Property	Provides the name of a folder in the compressed setup package or in the release location where this component's files are stored when you build a release. This property is identical to a component's Source Location setting in InstallShield.  ***Important*** • *Setting the Destination property for the ISWiComponent object resets the object's SourceLocation property. Therefore, any code that sets the SourceLocation property should be placed after any code that sets the Destination property.*

Table 12-41 • ISWiComponent Object Members (cont.)

Name	Project	Type	Description
Transitive	All	Read-Write Property	Set this property to True to have the installer re-evaluate the condition when this component is reinstalled, False otherwise.

Applies To

- ISWiFeature

AddComponentSubFolder Method



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The AddComponentSubFolder method adds a component subfolder to the current component or component subfolder.

Syntax

```
AddComponentSubFolder (Name As String) As ISWiComponentSubFolder
```

Parameters

Table 12-42 • AddComponentSubFolder Method

Parameter	Description
Name	Pass the name for the subfolder that you want to add.

Example

The following Visual Basic lines demonstrate the AddComponentSubFolder method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pComponent As ISWiComponent
Dim pComponentSubFolder As ISWiComponentSubFolder

Set pFeature = pProj.ISWiFeatures.Item("MyFeature")
Set pComponent = pFeature.ISWiComponents.Item("MyComp")
Set pComponentSubFolder = pComponent.AddComponentSubFolder("MyCompSubFolder")

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiComponent
- ISWiComponentSubFolder

AddDynamicFileLinking Method



Project *• The information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Call AddDynamicFileLinking to add a dynamic file link to a specified component. All of the files that are in the folder that you specify are automatically incorporated into your installation.



Project *• In Basic MSI, InstallScript MSI, and Merge Module projects, all new dynamic file links that you add use the best practice method of component creation by default. For more information, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.*

Syntax

```
AddDynamicFileLinking (sSourceFolder As String) As ISWiDynamicFileLinking
```

Parameters

Table 12-43 • AddDynamicFileLinking Method Parameters

Parameter	Description
sSourceFolder	Pass the fully qualified path to the folder that contains the files that you want to be dynamically linked.

Example

The following Visual Basic example adds a dynamic link to **MyDynamicFiles** to the component **MyComponent**:

```
m_ISWiFeature.ISWiComponents("MyComponent").AddDynamicFileLinking "C:\Build 141\MyDynamicFiles"
```

All of the files in the **MyDynamicFiles** folder are added to the project. The folder is scanned before every build, and any new or changed files are automatically incorporated into your project.

Applies To

- ISWiComponent

AddEnvironmentVar Method



Project • *The AddEnvironmentVar method applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Call AddEnvironmentVar to add an environment variable to a component.

Syntax

```
AddEnvironmentVar (sName As String) As ISWiEnvironmentVar
```

Parameters

Table 12-44 • AddEnvironmentVar Method Parameters

Parameter	Description
sName	Specifies the name of the environment variable.

Example

The following Visual Basic example adds the environment variable **MyEnvironment** to the component **MyComponent**:

```
m_ISWiFeature.ISWiComponents("MyComponent").AddEnvironmentVar "MyEnvironment"
```

Applies To

- ISWiComponent

AddFile Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Project-specific differences are noted where appropriate.

Call AddFile to add a file to a specified component or, in InstallScript projects, a component subfolder. The files that are added with this method have static file links.

Syntax

```
AddFile (FileName As String) As ISWiFile
```

Parameters

Table 12-45 • AddFile Method Parameters

Parameter	Description
FileName	Pass the fully qualified file name for the individual file that you want to add to this component or component subfolder.

Example

The following Visual Basic example adds the file MyFile.exe to the component MyComponent:

```
m_ISWiFeature.ISWiComponents("MyComponent").AddFile "C:\Build 141\MyFile.exe"
```

Applies To

- ISWiComponent
- ISWiComponentSubFolder

AddRemoveFile Method



Project • *The AddRemoveFile method applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The AddRemoveFile method adds a file or folder removal entry to the current component. This file and folder removal capability is useful for scenarios such as removing application-created files that your installation does not otherwise track.

Syntax

```
AddRemoveFile () As ISWiRemoveFile
```

Applies To

- ISWiComponent

DeviceDriverFlagsEx Property



Project • *This information applies to the following project types:*

Project-specific differences are noted where appropriate.

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The DeviceDriverFlagsEx property enables you to specify the flag settings for device drivers and runtime options directly from the automation layer. You can pass the following property values to this property. Pass more than one value by using OR.

Table 12-46 • DeviceDriverFlagsEx Property Values

Value	Definition
0	Not set (default).
1	To replace any existing driver for the device with this driver, pass this property value to the DeviceDriverFlagsEx property. If you do not pass this property and a driver already exists for the device, your driver is not installed.
2	To prevent the Connect Device to Computer dialog from being displayed during the installation of a device driver if a device matching the driver is not connected to a computer, pass this property value.
4	If you pass this property value to the DeviceDriverFlagsEx property, the installation creates an Add or Remove Programs entry for the device driver only; the installation does not create an additional entry for the installation itself. DIFxApp does not support this feature on Windows Vista and later.
8	By default, DIFxApp does not install unsigned driver packages or driver packages that have missing files. To override this default behavior, pass this value to the DeviceDriverFlagsEx property.
16	By default, when DIFxApp uninstalls a driver package, DIFxApp does not remove the binary files that were copied to the system when it installed the driver. To override this default behavior, pass this value to the DeviceDriverFlagsEx property. If you do pass this value, DIFxApp removes a binary file from the system only if the binary file is identical to the corresponding binary file in the driver store.  ***Caution*** • *Pass this value only if you can verify that the driver's binary files are not required by any other driver package or application.*
Other value	Invalid. This is a fatal installation error. DIFxApp does not install the component and uninstalls any components in the same installation package that were installed before this component.

Applies to

- ISWiComponent

DeviceDriverSetRedist Property



Project • *The DeviceDriverSetRedist property applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The DeviceDriverSetRedist property sets the custom action runtime .dll used for all device driver installations in the project. This property takes the following values:

- **eddr32Bit**—Specifies that the device drivers in this project target 32 bit machines and that the installation uses runtime dialogs for English only.
- **eddf32BitLocalized**—Specifies that the device drivers in this project target 32 bit machines and that the installation includes all localized installation runtime dialogs.
- **eddf64Bit**—Specifies that the device drivers in this project target Itanium 64 bit machines and that the installation uses runtime dialogs for English only.
- **eddf64BitLocalized**—Specifies that the device drivers in this project target Itanium 64 bit machines and that the installation includes all localized installation runtime dialogs.
- **eddf64BitAMD**—Specifies that the device drivers in this project target AMD 64 bit machines and that the installation uses runtime dialogs for English only.
- **eddf64BitAMDLocalized**—Specifies that the device drivers in this project target AMD 64 bit machines and that the installation includes all localized installation runtime dialogs.

The English runtime .dll file is smaller than the localized .dll file, so if space is an issue and you do not need the extra language runtime dialogs, use one of the English-only values (not one of the localized values).

If you use one of the localized values, the installation uses a runtime .dll that includes dialogs and text for the following languages:

- Arabic (Saudi Arabia)
- Spanish - Modern Sort (Spain)
- Norwegian (Bokmål) (Norway)
- Chinese (People's ROC)
- Finnish (Finland)
- Dutch (Netherlands)
- Chinese (Taiwan)
- French (France)
- Polish (Poland)
- Czech (Czech Republic)
- Hebrew (Israel)

- Portuguese (Brazil)
- Danish (Denmark)
- Hungarian (Hungary)
- Portuguese (Portugal)
- German (Germany)
- Italian (Italy)
- Russian (Russia)
- Greek (Greece)
- Japanese (Japan)
- Swedish (Sweden)
- English (United States)
- Korean (Korea)
- Turkish (Turkey)

Applies to

- ISWiComponent

ImportINIFile Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

Call the ImportINIFile method to import INI files from the automation layer.

Syntax

```
ImportINIFile(INIFile As String, [DirectoryID As String])
```

Parameters

Table 12-47 • ImportINIFile Method Parameters

Parameter	Description
INIFile	This represents the full path to the INI file that needs to be imported.
DirectoryID	The directory must be a valid MSI Directory identifier such as INSTALLDIR. The directory is optional if you do not specify it. InstallShield will use the component directory.

Applies To

- ISWiComponent

ImportRegFile Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the ImportRegFile method to import the contents of a registry (REG) file into the registry data for a specified component. This method mimics manually importing a REG file in the IDE.

Syntax

```
ImportRegFile (RegFile As String, Optional OverWrite As Boolean)
```

Parameters

Table 12-48 • ImportRegFile Method Parameters

Parameter	Description
RegFile	Specify the fully qualified path to the REG file that you want to import.
OverWrite	Choose one of the following options for instructing ImportRegFile how you want to handle conflicts when merging registry data: True—If a value exists in the component's registry data, overwrite it with the value from the REG file. False—If a value in the REG file is already present in the component's registry data, do not import that value. The default value for this optional parameter is True.

Applies To

- ISWiComponent

ISWiRemoveFiles Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The ISWiRemoveFiles method returns an ISWiRemoveFiles collection that contains all of the file and folder removal entries that are associated with the current component.

Syntax

```
ISWiRemoveFiles As ISWiRemoveFiles
```

Parameters

Table 12-49 • RemoveRemoveFile Method Parameters

Parameter	Description
pRemoveFolder	Pass the ISWiRemoveFile object that you want to remove.

Example

The following Visual Basic code illustrates how you would set a property for a specific file removal entry in a component. For the purposes of this example, assume that the variable m_oMyComp already has a reference to a component.

```
m_oMYComp.ISWiRemoveFiles("RemoveFile1").FileName = "*.txt"
```

Applies To

- ISWiComponent

RemoveComponentSubFolder Method



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The RemoveComponentSubFolder method removes the specified component subfolder from the current component or component subfolder.

Syntax

```
RemoveComponentSubFolder (pComponentSubFolder As ISWiComponentSubFolder) As Long
```

Parameters

Table 12-50 • RemoveComponentSubFolder Method Parameters

Parameter	Description
pComponentSubFolder	Pass the subfolder object that you want to remove.

Example

The following Visual Basic lines demonstrate the RemoveComponentSubFolder method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pComponent As ISWiComponent
Dim pComponentSubFolder As ISWiComponentSubFolder
```

```
Set pFeature = pProj.ISWiFeatures.Item("MyFeature")
Set pComponent = pFeature.ISWiComponents.Item("MyComp")
Set pComponentSubFolder = pComponent.ISWiComponentSubFolders.Item("MyCompSubFolder")
pComponent.RemoveComponentSubFolder pComponentSubFolder

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiComponent
- ISWiComponentSubFolder

RemoveDynamicFileLinking Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Call RemoveDynamicFileLinking to remove a dynamic file link from the current component.

Syntax

```
RemoveDynamicFileLinking (pSourceFolder As ISWiDynamicFileLinking) As Long
```

Parameters

Table 12-51 • RemoveDynamicFileLinking Method Parameters

Parameter	Description
pSourceFolder	Pass the ISWiDynamicFileLinking object to specify a dynamic file link that you want to remove from the project.

Applies To

- ISWiComponent

RemoveEnvironmentVar Method



Project • *This information applies to the following project types:*

- *Basic MSI*

- *DIM*
- *InstallScript MSI*
- *Merge Module*

Call RemoveEnvironmentVar to remove an environment from a specified component.

Syntax

```
RemoveEnvironmentVar (sName As ISWiRemoveEnvironmentVar) As Long
```

Parameters

Table 12-52 • RemoveEnvironmentVar Method Parameters

Parameter	Description
sName	Pass the ISWiEnvironmentVar object to specify an environment variable that you want to remove.

Applies To

- ISWiComponent

RemoveFile Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call RemoveFile to remove a file from a specified component.

Syntax

```
RemoveFile (pFile As ISWiFile) As Long
```

Parameters

Table 12-53 • RemoveFile Method Parameters

Parameter	Description
pFile	Pass the ISWiFile object you want to remove.

Applies To

- ISWiComponent

RemoveRemoveFile Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The RemoveRemoveFile method removes a file or folder removal entry from the current component. To remove a file or folder removal entry from the project, pass the ISWiRemoveFile object.

Syntax

```
RemoveRemoveFile (pRemoveFolder As ISWiRemoveFile) As Long
```

Parameters

Table 12-54 • RemoveRemoveFile Method Parameters

Parameter	Description
pRemoveFolder	Pass the ISWiRemoveFile object that you want to remove.

Applies To

- ISWiComponent

ISWiComponentSubFolder Object



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

The ISWiComponentSubFolder object represents a component subfolder belonging to the current component or component subfolder. The component subfolder can be an existing one from the InstallShield user interface's Components view or one added by calling AddComponentSubFolder.

Use the project's ISWiComponentSubFolders collection to access a specific component subfolder. You must specify either an index number or the component subfolder name for the Item property of ISWiComponentSubFolders.

Members

Table 12-55 • ISWiComponentSubFolder Object Members

Name	Type	Description
AddComponentSubFolder	Method	Adds a subfolder to the component subfolder.
AddFile	Method	Adds a file to the component subfolder.
DeleteFile	Method	Deletes a file from the component subfolder.
ISWiComponentSubFolders	Collection	Contains all of the subsubfolders associated with this component subfolders.
ISWiFiles	Collection	Contains all of the files associated with this component subfolder.
Name	Read-Only Property	Stores the name of the component subfolder.
RemoveComponentSubFolder	Method	Removes a subfolder from the component subfolder.

Example

The following Visual Basic code illustrates the use of the ISWiComponentSubFolder object:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pComponent As ISWiComponent
Dim pComponentSubFolder As ISWiComponentSubFolder

Set pFeature = pProj.ISWiFeatures.Item("MyFeature")
Set pComponent = pFeature.ISWiComponents.Item("MyComp")
Set pComponentSubFolder = pComponent.ISWiComponentSubFolders.Item("MyCompSubFolder")
pComponentSubFolder.AddFile("C:\My Files\MyFile.ext")

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiComponent

DeleteFile Method



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

Call DeleteFile to delete a file from a specified component subfolder. The files added with this method have static file links, similar to adding files in the IDE.

Syntax

```
DeleteFile (pFile As ISWiFile) As Long
```

Parameters

Table 12-56 • DeleteFile Method Parameters

Parameter	Description
pFile	Pass the file object that you want to delete.

Example

The following Visual Basic example deletes a file from the component subfolder MyCompSubFolder:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pComponent As ISWiComponent
Dim pComponentSubFolder As ISWiComponentSubFolder
Dim pFile As ISWiFile

Set pFeature = pProj.ISWiFeatures.Item("MyFeature")
Set pComponent = pFeature.ISWiComponents.Item("MyComp")
Set pComponentSubFolder = pComponent.ISWiComponentSubFolders.Item("MyCompSubFolder")
Set pFile = pComponentSubFolder.ISWiFiles.Item("MyFile.ext")
pComponentSubFolder.DeleteFile pFile

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiComponentSubFolder

ISWiCondition Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

ISWiCondition is a subset of the ISWiFeature object. This object allows you to get and set conditional logic for any of your project's features.

Members

Table 12-57 • ISWiCondition Object Members

Name	Type	Description
Condition	Read-Write Property	Gets or sets any conditions that are associated with this feature.
Level	Read-Write Property	Gets or sets the install level assigned to the feature when the condition evaluates to TRUE.

Applies To

- ISWiFeature

ISWiDynamicFileLinking Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Project-specific differences are noted where appropriate.

ISWiDynamicFileLinking represents a dynamic file link that belongs to a component. The dynamic file link can be an existing one from the InstallShield user interface or one added by calling AddDynamicFileLinking.

Members

Table 12-58 • ISWiDynamicFileLinking Object Members

Name	Project	Type	Description
CreateBestPracticeComponents	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	To use the best practice method of component creation for your dynamically linked files, set this property to True. To use the one-component-per-directory method of component creation, set this property to False. For detailed information about the two methods, see Determining the Appropriate Component Creation Method for Dynamically Linked Files.
DynamicSubfolders	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Set this property to True to include all subfolders that are available in SourceFolder.
ExcludeFiles	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Specify the file types that you want to exclude. Enter the file extension preceded by an asterisk (*). Separate multiple entries with a comma.
IncludeFiles	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Specify the file types that you want to include. Enter the file extension preceded by an asterisk (*). Separate multiple entries with a comma.
SelfRegister	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Set this property to True to self-register every file in the dynamically linked folder. If the Attrib64BitComponent property of the component is set to True, the installation performs 64-bit self-registration on the target system. For more information, see Targeting 64-Bit Operating Systems.
SourceFolder	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Specifies the fully qualified path to the folder whose contents should be dynamically linked.

Applies To

- ISWiComponent

ISWiEnvironmentVar Object



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

ISWiEnvironmentVar represents an environment variable that belongs to a component. The file can be an existing one from the InstallShield user interface or one added by calling AddEnvironmentVar.

Members

Table 12-59 • ISWiEnvironmentVar Object Members

Name	Type	Description
EnvType	Read-Write Property	This enumerated integer property specifies whether the environment variable name refers to a system or user environment variable: • **evtSystem**—Creates, modifies, or removes the specified system environment variable. • **evtUser**—Creates, modifies, or removes the environment variable from the user's environment. The specified environment variable is available only to the end user who is logged on at the time of installation.
Key	Read-Only Property	Stores the key of the **Environment** table.
Name	Read-Write Property	Retrieves or sets the name of the environment variable.
OnInstall	Read-Write Property	This enumerated integer property retrieves or sets the action that you want to occur when the associated feature is installed to the target system: • **evoiSet**—In conjunction with the Placement property, sets the value of an existing environment variable. This option creates the environment variable if it does not already exist on the target system, and then it sets it during installation. If the environment variable exists on the target system, it is set during the installation. • **evoiCreate**—Creates the specified environment variable on the target system if it does not already exist, and it sets the variable's value. • **evoiRemove**—Removes the specified environment variable from the target system.
OnUninstall	Read-Write Property	This enumerated integer property specifies whether this environment variable should be removed from the target system when the associated feature is uninstalled: • **evouRemove**—Removes the environment variable or its appended value from the target system if the associated feature is uninstalled. If the value of the OnInstall property is Create, the Remove value for OnUninstall removes the entire environment variable. If the value of the OnInstall property is Set, the Remove value for OnUninstall removes only the value that was appended to the variable's value. • **evouLeave**—Leaves the environment variable or its appended value on the target system if the associated feature is uninstalled.

Table 12-59 • ISWiEnvironmentVar Object Members (cont.)

Name	Type	Description
Placement	Read-Write Property	This enumerated integer property retrieves or sets the placement of the value in the Value field relative to the specified environment variable's existing value: • **evpAppend**—Appends the value indicated in the Value property to the end of the specified environment variable's existing value. • **evpPrefix**—Adds the value indicated in the Value property to the beginning of the specified environment variable's existing value. • **evpReplace**—Replaces the value of the specified environment variable with the value indicated in the Value property.
Value	Read-Write Property	Retrieves or sets the path or value for this environment variable.

Applies To

- ISWiComponent

ISWiFeature Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Project-specific differences are noted where appropriate.

ISWiFeature represents a top-level feature or a subfeature in InstallShield. You can retrieve a feature by specifying an item in the ISWiFeatures collection.

Many feature properties and attributes are available in the automation interface through this object's methods, properties, and collections. Refer to the following table to see which methods, properties, and collections are available for your project type.

Members

Table 12-60 • ISWiFeature Object Members

Name	Project	Type	Description
AddCondition	Basic MSI, InstallScript MSI	Method	Adds a condition to the current ISWiFeature.
AddFeature	Basic MSI, InstallScript, InstallScript MSI	Method	Creates an ISWiFeature object as a subfeature of the current ISWiFeature.
AddMergeModule	Basic MSI, InstallScript MSI	Method	Associates a merge module with the current feature.
AddObject	InstallScript	Method	Adds an InstallScript object to the current ISWiFeature.
Advertise	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the Advertisement property for the current feature. Use the following constants to set this property: • **eaAllow (1)**—Use this option to enable advertisement on this feature. Although advertisement is allowed, it is not the default option when the setup is run. • **eaFavor (2)**—Use this option to have the feature advertised by default. Your end users can always change the advertisement option for a feature in Custom Setup dialog. • **eaDisallow (3)**—Use this option if you do not want to allow advertising for this feature. Your end users will not be able to choose to have a feature advertised in the Custom Setup dialog. • **eaDisableIfNotSupported (4)**—Advertisement only works on systems with Internet Explorer 4.01 or higher. If the target system does not meet this criteria, advertising is turned off. If the target system can support advertisement, advertising is allowed.
AttachComponent	Basic MSI, InstallScript, InstallScript MSI	Method	Associates a component with the current feature.

Table 12-60 • ISWiFeature Object Members (cont.)

Name	Project	Type	Description
CDRomFolder	InstallScript	Read-Write Property	Gets or sets the feature's CD-ROM Folder property, which specifies the folder in the disk image in which the feature is placed if the feature's check box is checked in the Release wizard's Custom Media Layout panel.
Comments	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the comments associated with this feature.
DeleteCondition	Basic MSI, InstallScript MSI	Method	Deletes a condition associated with the current feature.
Description	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the description for this feature.
DescriptionID	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the string identifier for this feature's description. ***Note*** • *To set this property, the string entry must exist and you must know the string identifier.*
Destination	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the default destination of this feature.
Display	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the Display property for this feature. Use the following constants to set this property: • **edtVisibleAndCollapsed (0)**—Means that the feature is displayed in the Custom Setup dialog with its subfeatures collapsed by default. • **edtVisibleAndExpanded (1)**—Means that the feature is displayed in the Custom Setup dialog with its subfeatures expanded by default. • **edtNotVisible (2)**—Means that the feature is not displayed to the end user in the Custom Setup dialog.

Table 12-60 • ISWiFeature Object Members (cont.)

Name	Project	Type	Description
DisplayName	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the display name of this feature.
DisplayNameID	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the string identifier for this feature's display name. ***Note*** • *To set this property, the string entry must exist and you must know the string identifier.*
Encryption	InstallScript	Read-Write Property	Gets or sets the feature's Encryption property, which specifies whether the files for the feature will be encrypted by the release builder. This property is a Boolean.
Features	Basic MSI, InstallScript, InstallScript MSI	Read-Only Property	This property contains an ISWiFeatures collection of all this feature's immediate subfeatures, if any.
FTPLocation	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the feature's FTP Location property, which specifies an FTP location for the feature.
HTTPLocation	InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the feature's HTTP Location property, which specifies an HTTP location for the feature.
IncludeInBuild	InstallScript	Read-Write Property	Gets or sets the feature's Include In Build property, which specifies whether the feature should be included in your distribution media. This property is a Boolean.
GUID	InstallScript	Read-Write Property	Gets or sets the feature's GUID property, which identifies the feature in the setup's log file; if you change a feature's GUID in an update setup, the setup engine treats it as a new feature rather than an update to an existing feature.
InstallLevel	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the install level of this feature.

Table 12-60 • ISWiFeature Object Members (cont.)

Name	Project	Type	Description
ISWiComponents	Basic MSI, InstallScript, InstallScript MSI	Collection	Contains all of the components associated with the current feature.
ISWiConditions	Basic MSI, InstallScript MSI	Collection	Contains all of the conditions associated with the current feature.
ISWiObjects	InstallScript	Collection	Contains all of the InstallScript objects associated with the current feature.
Miscellaneous	InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the feature's Miscellaneous property, which associates a text string with the feature.
Name	Basic MSI, InstallScript, InstallScript MSI	Read-Only Property	Feature name of the current element in the collection.
Need	InstallScript	Read-Write Property	Gets or sets the feature's File Need property, which specifies whether a warning, a severe warning, or no warning should be given to end users who deselect the feature. Use the following constants to set this property: • **efnCritical (0)**—Use this option for features that your setup does not allow end users to deselect. • **efnHighlyRecommended (1)**—Use this option to advise end users that the feature should be installed. • **efnStandard (2)**—Use this option to let end users deselect the feature with no message issued.
OnInstalled	InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the feature's OnInstalled property, which specifies the function to be executed just after the feature is installed.
OnInstalling	InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the feature's OnInstalling property, which specifies the function to be executed just before the feature is installed.
OnUninstalled	InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the feature's OnUninstalled property, which specifies the function to be executed just after the feature is uninstalled.

Table 12-60 • ISWiFeature Object Members (cont.)

Name	Project	Type	Description
OnUninstalling	InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the feature's OnUninstalling property, which specifies the function to be executed just before the feature is uninstalled.
Password	InstallScript	Read-Write Property	Gets or sets the feature's Password property, which specifies a password for the feature.
ReleaseFlags	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the release flags that are associated with this feature. Separate multiple flags with a comma. You can specify which release flags you want to use to filter the project's features in the ReleaseFlags property of ISWiRelease.
RemoteInstallation	Basic MSI, InstallScript MSI	Read-Write Property	Retrieves or sets the feature's Remote Installation property. Use the following constants to set this property: • **efrLocal (0)**—Means that all of the feature's files are installed onto the target system. • **efrSource (1)**—Causes the files belonging to this feature to run directly from the source medium, such as a CD-ROM or network server. • **efrFollowParent (2)**—Gives a subfeature the Remote Installation property of its parent feature.
RemoveComponent	Basic MSI, InstallScript, InstallScript MSI	Method	Removes a component from this feature.
RemoveFeature	Basic MSI, InstallScript, InstallScript MSI	Method	Deletes the specified feature from the project.
RemoveMergeModule	Basic MSI, InstallScript MSI	Method	Removes a merge module from the current feature.
RemoveObject	InstallScript	Method	Removes an InstallScript object from the current ISWiFeature.
Required	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the feature's Required property. This property is a Boolean.

Table 12-60 • ISWiFeature Object Members (cont.)

Name	Project	Type	Description
RequiredFeatures	InstallScript, InstallScript MSI	Read-Write Property	Gets or sets a comma-delimited string that lists the names of the features that are specified in the current feature's Required Features property.

SetupTypeStatus	InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the selection status of the feature in the setup type that is specified in the property's argument. Use the following constants to set this property:
			• **estSelected (10)**—The feature is selected.
			• **estDeselected (11)**—The feature is not selected.
			In the following Visual Basic code snippet, the Help_Files feature is not selected for the Compact setup type.
			`Set pFeature = pProject.ISWiFeatures.Item("Help_Files")`
			`Set pSetupType = pProject.ISWiSetupTypes.Item("Compact")`
			`pFeature.SetupTypeStatus(pSetupType) = 11`

StatusText	InstallScript	Read-Write Property	Gets or sets the feature's Status Text property, which specifies the text that end users see in the uppermost line of the progress indicator during the transfer of this feature.
Visible	InstallScript	Read-Write Property	Gets or sets the feature's Visible property, which specifies whether the feature will be visible in feature dialog boxes during the installation. This property is a Boolean.

Applies To

- ISWiProject

AddCondition Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the AddCondition method to add a condition to the current feature.

Syntax

```
AddCondition (sKey As String) As ISWiCondition
```

Parameters

Table 12-61 • AddCondition Method Parameters

Parameter	Description
sKey	Pass the string version of the condition you want to attach to the current feature.

Applies To

- ISWiFeature

AddMergeModule Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the AddMergeModule method to associate a merge module with the current feature. This method is similar to adding a merge module to a feature in the Redistributables view.

Syntax

```
AddMergeModule (FileName As String)
```

Parameters

Table 12-62 • AddMergeModule Method Parameters

Parameter	Description
FileName	Pass the fully qualified file name for the merge module (.msm) file that you want to associate with this feature.

Applies To

- ISWiFeature

AddObject Method



Project • *This information applies to InstallScript projects.*

The AddObject method adds an InstallScript object with the specified moniker to the current feature.

Syntax

```
AddObject (Moniker As String) As ISWiObject
```

Parameters

Table 12-63 • AddObject Method Parameters

Parameter	Description
Moniker	Pass the moniker for the InstallScript object that you want to add. To get an InstallScript object's moniker, create a new Professional project in the IDE and add the InstallScript object to a feature, then go to the Direct Editor view's ISFeatureExtended table and see the Moniker column.

Example

The following Visual Basic lines demonstrate the AddObject method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pObject As ISWiObject

Set pFeature = pProj.ISWiFeatures.Item(2)
Set pObject = pFeature.AddObject ("@ismk2:755142B0-1EB4-11D3-8B09-00105A9846E9")

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiFeature

AttachComponent Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Call the AttachComponent method to associate a component with the current feature. The component must exist (created manually in the IDE or by calling AddComponent) before you can associate it with a feature.

Syntax

```
AttachComponent (Component As ISWiComponent)
```

Parameters

Table 12-64 • AttachComponent Method Parameters

Parameter	Description
Component	Provide the ISWiComponent object that represents the component you want to associate with this feature.

Applies To

- ISWiFeature

DeleteCondition Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the DeleteCondition method to delete a condition from the current feature.

Syntax

```
DeleteCondition (pCondition As ISWiCondition)
```

Parameters

Table 12-65 • DeleteCondition Method Parameters

Parameter	Description
pCondition	The ISWiCondition object representing the condition you want to delete.

Applies To

- ISWiFeature

RemoveComponent Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

Call RemoveComponent to remove a component from a specified feature. Similar to removing a component in the Setup Design views, RemoveComponent disassociates a component from its feature, but it does not permanently delete it from the project.

Syntax

```
RemoveComponent (pComponent As ISWiComponent) As Boolean
```

Parameters

Table 12-66 • RemoveComponent Method Parameters

Parameter	Description
pComponent	Enter the ISWiComponent object that you want to remove.

Applies To

- ISWiFeature

RemoveMergeModule Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the RemoveMergeModule method to remove a merge module from only the current feature. (To delete a merge module from the project, call the DeleteMergeModule method.)

Syntax

```
RemoveMergeModule (FileName As String)
```

Parameters

Table 12-67 • RemoveMergeModule Method Parameters

Parameter	Description
FileName	Pass the fully qualified file name for the merge module (.msm) file that you want to remove from this feature.

Applies To

- ISWiFeature

RemoveObject Method



Project • *This information applies to InstallScript projects.*

The RemoveObject method removes the specified InstallScript object from the current feature.

Syntax

```
RemoveObject (Object As ISWiObject) As Long
```

Parameters

Table 12-68 • RemoveObject Method Parameters

Parameter	Description
Object	The InstallScript object to be deleted.

Example

The following Visual Basic lines demonstrate the RemoveObject method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pObject As ISWiObject

Set pFeature = pProj.ISWiFeatures.Item("Help_Files")
Set pObject = pFeature.ISWiObjects.Item(1)
pFeature.RemoveObject pObject

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiFeature

ISWiFile Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Project-specific differences are noted where appropriate.

ISWiFile represents a file belonging to a component or, in InstallScript projects, a component subfolder in your project. The file can be an existing one from the InstallShield user interface or one added by calling AddFile.

Many feature properties and attributes are available in the automation interface through this object's methods, properties, and collections. Note that some methods, properties, and collections are available for specific project types.

Use the component's ISWiFiles collection to access a specific file. You must specify either an index number or the file key for the Item property of ISWiFiles. The file key is visible in the **File** column of the **File** table in the Director Editor view of InstallShield. In Basic MSI and InstallScript MSI projects, it is also listed under the **Key** column in a component's file list.

Unfortunately, you might not know the file key, or it might have been reassigned if the files are dynamically linked to the component. Instead of using the file key, you could write code similar to the following example to check for a certain file name:

```
Dim m_pFile As ISWiFile

For Each m_pFile In m_pComponent.ISWiFiles
    If UCase(m_pFile.DisplayName) = "MYFILE.EXE" Then
        Exit For
    End If
Next
```

If the file name MyFile.exe is found among the component's files, the For loop ends, and m_pFile holds a copy of the ISWiFile object for MyFile.exe.

Members

Table 12-69 • ISWiFile Object Members

Name	Project	Type	Description
Attributes	Basic MSI, InstallScript MSI	Property (Deprecated)	This property is no longer supported. Use the Hidden, ReadOnly, System, and Vital properties instead.
CompanionFile	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Reserved for future use.
DisplayName	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	The standard name of the file. Be sure that case of this entry matches the case of the file name if you plan on distributing your package via the Internet, since some Web servers are case-sensitive. The name cannot contain any characters that are not valid for Windows folders and file names. If the file has both a long and a short file name (see ShortName, below), only the long name is used in the display name. You cannot modify this property if the files are dynamically linked.
DynamicFile	Basic MSI, DIM, InstallScript, InstallScript MSI, Merge Module	Read-Only Property	This property is True if the file's source is linked to the component dynamically, or False if it is linked statically.
FontTitle	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	Retrieves or sets the Font Title file attribute as a string. You cannot modify this property if the files are dynamically linked.
FullPath	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Stores the fully qualified path (including the file name) of the file's source location. Any path variables are resolved to the actual path. You cannot modify this property if the files are dynamically linked.

Table 12-69 • ISWiFile Object Members (cont.)

Name	Project	Type	Description
Hidden	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	True if a file has the Hidden attribute set. This property is ignored if OverrideSystemAttributes is set to False. You cannot modify this property if the files are dynamically linked.
Languages	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	This property should contain a string of all of the file's language IDs, in decimal and separated by commas. The following line identifies the file as English and German: m_pFile.Languages = "1031,1033" This property is ignored if OverrideSystemLanguage is set to True. You cannot modify this property if the files are dynamically linked.
Modified	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	Retrieves the file's Modified file attribute. Reserved for future use.
Name	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	Retrieves or sets a file key. See the description of ISWiFile above for more information about the file key. You cannot modify this property if the files are dynamically linked.
OverrideSystemAttributes	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	To override the development system's settings for all of the file attributes, such as Hidden, ReadOnly, Vital, and System, set this property to True. If you set this property to False, the Hidden, ReadOnly, Vital, and System properties are ignored. You cannot modify this property if the files are dynamically linked.
OverrideSystemLanguage	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	To override the language of the file on the development system, set this property to True. If you set this property to False, the Languages property is ignored. You cannot modify this property if the files are dynamically linked.

Table 12-69 • ISWiFile Object Members (cont.)

Name	Project	Type	Description
OverrideSystemSize	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	To override the size of the file on the development system, set this property to True. If you set this property to False, the Size property is ignored. You cannot modify this property if the files are dynamically linked.
OverrideSystemVersion	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	To override the version of the file version on the development system, set this property to True. If you set this property to False, the Version property is ignored. You cannot modify this property if the files are dynamically linked.
ReadOnly	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Retrieves or sets the Read-only file attribute. Set this property to True to mark a file as read-only. Leave it undefined or set it to False to mark a file as read-write. This property is ignored if OverrideSystemAttributes is set to False. You cannot modify this property if the files are dynamically linked.
SelfRegister	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the Self-Register file attribute. Set this property to True to mark a file as self-registering. You cannot modify this property if the files are dynamically linked.

Table 12-69 • ISWiFile Object Members (cont.)

Name	Project	Type	Description
ShortName	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Gets or sets the file's short name. Your files must have corresponding short names in case the Windows Installer property SHORTFILENAMES is set because the distribution medium cannot handle long file names. Only the short file names are used when you deselect "Use long file names" in the Advanced Settings panel of the Release wizard. By default, the build process will generate short names for your files, and therefore reading the current value of a file's ShortName property will return an empty string. However, if you explicitly set a file's ShortName property, the build process will use the short name you specify, and reading the ShortName value will return the short name you specified. You cannot modify this property if the files are dynamically linked.
Size	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	Contains the size of the file, in bytes, as a Long. If you are setting this property, OverrideSystemSize must first be set to True. Otherwise, InstallShield uses the actual size of the file. You cannot modify this property if the files are dynamically linked.
System	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	Retrieves or sets the System file attribute. Set this property to True if the file is a system file. This property is ignored if OverrideSystemAttributes is set to False. You cannot modify this property if the files are dynamically linked.
UseSystemSettings	Basic MSI, InstallScript MSI	Property (Obsolete)	This property is no longer supported. Use the OverrideSystemAttributes, OverrideSystemSize, OverrideSystemVersion, and OverrideSystemLanguage properties instead.

Table 12-69 • ISWiFile Object Members (cont.)

Name	Project	Type	Description
Version	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	Set this property to the version of the file that you are including in the setup. To specify a companion file, this property can also be set to the Name property of another ISWiFile object, provided. To learn more about companion files, see Companion Files. This property is ignored if OverrideSystemVersion is set to False. You cannot modify this property if the files are dynamically linked.
Vital	Basic MSI, DIM, InstallScript MSI, Merge Module	Read-Write Property	Retrieves or sets the Vital file attribute. Set this property to True to mark this file as vital. You cannot modify this property if the files are dynamically linked.

ISWiFolder Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

ISWiFolder represents a program folder in a component's Shortcuts explorer in the InstallShield user interface. You can retrieve a folder by specifying an item in the ISWiFolders collection.

Many folder attributes are available in the automation interface through this object's methods and properties. Refer to the following table to learn which methods, properties, and collections are available for your project type.

Members

Table 12-70 • ISWiFolder Object Members

Name	Type	Description
AddShortcut	Method	Adds a shortcut object to the current folder object.
AddSubFolder	Method	Creates another program folder under the current one.
DeleteShortcut	Method	Deletes the specified shortcut from the current folder object.
DeleteSubFolder	Method	Deletes the specified subfolder from the current one.
Description	Read-Write Property	Contains any descriptive text you have entered for the folder.
DisplayName	Read-Write Property	Retrieves or sets the name displayed for a folder.
ISWiShortcuts	Collection	Contains every shortcut under the current program folder.
Name	Read-Write Property	Retrieves the key name of the folder. This is the name that is displayed in the Shortcuts view in the InstallShield user interface. It is not the same as the DisplayName property.
SharedFolder	Read-Write Property	This Boolean property retrieves or sets the Shared property of the folder, which specifies whether the folder is always removed from the system (False) or is removed only if it has no subfolders or files that were not created by the setup (True).
SubFolders	Collection	Contains an ISWiFolders collection of each program folder directly under the current folder as laid out in the Shortcuts explorer. See ISWiFolders Collection for an example of using this property to retrieve a subfolder. ***Note*** • *This property does not apply to InstallScript projects.*

Applies To

- ISWiComponent

AddShortcut Method

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Call AddShortcut to create a shortcut under the current program folder.

Syntax

```
AddShortcut (Name As String) As ISWiShortcut
```

Parameters

Table 12-71 • AddShortcut Method Parameters

Parameter	Description
Name	Enter the name you want to give the shortcut in the Shortcuts explorer.

Applies To

- ISWiFolder

AddSubFolder Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the AddSubFolder method to create a program folder directly under the current ISWiFolder object.

Syntax

```
AddSubFolder (Name As String) As ISWiFolder
```

Parameters

Table 12-72 • AddSubFolder Method Parameters

Parameter	Description
Name	Enter the name you want to give the program folder in the Shortcuts explorer.

Applies To

- ISWiFolder

DeleteShortcut Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Call DeleteShortcut to create a shortcut from the current folder.

Syntax

```
DeleteShortcut (pShortcut As ISWiShortcut) As Long
```

Parameters

Table 12-73 • DeleteShortcut Method Parameters

Parameter	Description
pShortcut	Pass the ISWiShortcut Object to specify a shortcut that you want to delete from a folder.

Examples

```
Set pShortcut = pFolder.ISWiShortcuts("MyShortcut")
If Not pShortcut Is Nothing Then
   pFolder.DeleteShortcut pShortcut
Next
```

Applies To

- ISWiFolder

DeleteSubFolder Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the DeleteSubFolder method to delete the specified subfolder from the current ISWiFolder object.

Syntax

```
DeleteSubFolder (pSubFolder As ISWiFolder) As Long
```

Parameters

Table 12-74 • DeleteSubFolder Method Parameters

Parameter	Description
Name	Pass the ISWiFolder Object to specify a subfolder that you want to delete from a folder.

Example

```
Set pSubFolder = pFolder.SubFolders("MySubFolder")
If Not pSubFolder Is Nothing Then
    pFolder.DeleteSubFolder pSubFolder
Next
```

Applies To

- ISWiFolder

ISWiLanguage Object



Edition • *Support for multilanguage installations is available with InstallShield Premier Edition.*



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

ISWiLanguage represents a language whose string entries are included in the current project.

To include or exclude languages, use the IsIncluded property on each relevant ISWiLanguage object.

Members

Table 12-75 • ISWiLanguage Object Members

Name	Type	Description
AddStringEntry	Method	Adds a new string entry to a language.
DeleteStringEntry	Method	Removes a string entry from the project.
Id	Read-Only Property	Stores the language identifier of the current language.
IsIncluded	Read-Write Property	Includes or excludes a language, or checks its current status.
ISWiStringEntries	Collection	Contains all of the string entries in the current language.
Name	Read-Only Property	Gets the name of the current language.

Applies To

- ISWiProject

AddStringEntry Method

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call AddStringEntry to add a new string entry to a language and return ISWiStringEntry.

Syntax

```
AddStringEntry (sLangId) As ISWiStringEntry
```

Parameter

Table 12-76 • AddStringEntry Method Parameter

Parameter	Description
sLangId	Specify the identifier of the language that should contain the string entry.

Applies To

- ISWiLanguage

DeleteStringEntry Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call DeleteStringEntry to delete a string entry from a language.

Syntax

```
DeleteStringEntry (pLangId As ISWiStringEntry)
```

Parameter

Table 12-77 • DeleteStringEntry Method Parameter

Parameter	Description
pLangId	Specify the ISWiStringEntry object to specify a string entry that you want to remove from the project.

Applies To

- ISWiLanguage

ISWiObject Object



Project • *This information applies to the following project types:*

- *InstallScript*

- *InstallScript Object*

The ISWiObject object represents an InstallScript object belonging to your project. The InstallScript object can be an existing one from the InstallShield user interface's Objects view or one added by calling AddObject.

Use the project's ISWiObjects collection to access a specific InstallScript object. You must specify either an index number or the moniker for the Item property of ISWiObjects. The moniker is visible in the IDE only as the Moniker in the Direct Editor view's **ISFeatureExtended** table.

Members

None.

Example

The following Visual Basic code illustrates the use of the ISWiObject:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pObject As ISWiObject

Set pFeature = pProj.ISWiFeatures.Item(2)
Set pObject = pFeature.ISWiObjects.Item("@ismk2:755142B0-1EB4-11D3-8B09-00105A9846E9")
pFeature.RemoveObject pObject

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiFeature

ISWiPathVariable Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

ISWiPathVariable represents a path variable that is included in the current project. The path variable can be an existing one that was created from the Path Variables view in InstallShield, or one that was added by calling the AddPathVariable method.



Important *• Predefined path variables cannot be modified or deleted. If you attempt to do so, exception error 3142 occurs. For more information, see Predefined Path Variables.*

Members

Table 12-78 • ISWiPathVariable Object Members

Name	Type	Description
Name	Read-Write Property	Retrieves or sets the name of the path variable.
PathVarType	Read-Write Property	This enumerated integer property specifies the type of the path variable. Available values are: • **epvtPreDefined (1)**—Predefined path variables are variables that are set to certain standard Windows directories. These variables cannot be renamed or modified, but they can be used in your installation project to point to predefined directories. • **epvtCustom (2)**—Custom path variables are sometimes referred to as *standard path variables* or *user-defined variables*. These variable types are defined through InstallShield. These variables do not rely on any outside sources, such as the registry or system paths. • **epvtEnvironment (4)**—Environment path variables are based on the values of your system's environment variables. You can set an environment path variable in your project to an existing environment variable on your machine. • **epvtRegistry (8)**—Registry path variables enable you to define your own variables based on the default value of a specified registry key. The registry key must already be present when you set a path variable to the value of the key.
TestValue	Read-Write Property	Retrieves or sets the test value of the path variable.
Value	Read-Write Property	Retrieves or sets the value of the path variable. **For Standard Path Variables** For standard variables, enter the directory to which you would like your variable to point.  ***Note*** *• You can also refer to other path variables in the defined value by enclosing the referenced path variable name in angled brackets. For example, if you have a path variable called* `MyRoot` *with a value of* **C:** *you can refer to it in a path variable definition for another variable, such as* `Games`*. The actual path for the* `Games` *variable might be* **C:\Programs\GameFiles,** *but you can define* `Games` *as* **<MyRoot>\Programs\GameFiles***. However, if you attempt to self-reference a path variable, the literal string is used instead. For example, defining* `Games` *as* **<MyRoot>\Programs\<Games>** *actually results in* `Games` *defined as* **C:\Programs\<Games>**.

Table 12-78 • ISWiPathVariable Object Members (cont.)

Name	Type	Description

Value (cont.)		**For Registry Path Variables**
		
		Enter the complete registry key, with the final "subkey" being the name of the value whose data contains the folder. For example, define *MyRegVar* as follows:
		
		`HKEY_LOCAL_MACHINE\SOFTWARE\TestKey\TestValue`
		
		Assume that TestKey has the following subkey and values:
		
		`[HKEY_LOCAL_MACHINE\SOFTWARE\TestKey]`
		`@="C:\\MyPath1"`
		`"TestValue"="C:\\MyPath2"`
		`[HKEY_LOCAL_MACHINE\Software\TestKey\TestValue]`
		`@="C:\\MyPath3"`
		
		Even though HKEY_LOCAL_MACHINE\SOFTWARE\TestKey has a subkey called TestValue, *MyRegVar* points to the value TestValue, and the current value will be **C:\MyPath2**. (Note, however, that if a value named TestValue does not exist, InstallShield reads the default value (**C:\MyPath3**) of the subkey HKEY_LOCAL_MACHINE\SOFTWARE\TestKey\TestValue.)
		
		For Environment Path Variables
		
		For environment path variables, enter the name of the variable as it appears on the Environment dialog box.

Applies To

- ISWiProject

ISWiProductConfig Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

InstallScript and InstallScript Object projects contain only one element in ISWiProductConfigs; it is named Media.

Only some of the members of ISWiProductConfig apply to InstallScript and InstallScript Object projects.

ISWiProductConfig represents a product configuration in the Releases view of the InstallShield user interface. You can retrieve a configuration by specifying an item in the ISWiProductConfigs collection.

Members

Table 12-79 • ISWiProductConfig Object Members

Name	Project	Type	Description
AddRelease	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Adds a release to the current product configuration.
ArchitectureValidation	Basic MSI, Merge Module	Read-Write Property	Set this property to one of the following values to indicate what type of build-time architecture validation you want to perform: • **epvtLenient (0)**—Lenient architecture validation, which is the default type of validation, does not trigger build errors or warnings if the architecture that the Template Summary property specifies does not match the architecture for one or more of the product files or the custom action files that are being included in the release. • **epvtStrict (1)**—Strict validation may trigger build errors and warnings if the architecture that the Template Summary property specifies does not match the architecture for one or more of the product files or custom action files that are being included in the release. The value of this property also influences whether InstallShield includes 32-bit or 64-bit versions of the built-in InstallShield custom action DLLs. To learn more, see Selecting the Appropriate Type of Architecture Validation for Builds.
ConfigFlags	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets a string that contains the release flags that you want to use to filter your features. Separate multiple flags with a comma.

Table 12-79 • ISWiProductConfig Object Members (cont.)

Name	Project	Type	Description
DeleteRelease	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Deletes the specified release from the current product configuration.
GeneratePackageCode	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Set this property to True to generate a new package code for every release you build under this product configuration. If this property is not specified or set to False, then the existing package code under the product configuration is used. However, the Summary Information Stream's package code is used if you have not specified one here.
ISWiReleases	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Collection	Contains all of the releases belonging to this product configuration.
MSIPackageFileName	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the value of the MSI Package File Name setting of the current product configuration.
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Only Property	Returns the name of the current product configuration.

Table 12-79 • ISWiProductConfig Object Members (cont.)

Name	Project	Type	Description
PackageCode	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets a string GUID for the package code that is built into this product configuration's releases. Neither the package code you enter in this property nor the Summary Information Stream's package code is used if you set the GeneratePackageCode property, above, to True.
ProductCode	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets a string GUID for the product code that is built into this product configuration's releases. ***Note*** • *The GUID must be surrounded by braces{12345678-1234-1234-1234-1234567890AB}, for example.*
ProductName	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the product name that is built into this product configuration's releases.
ProductVersion	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets a string that contains the product's version number, which overrides the Product Version property for every release you build under this product configuration.
SetupFileName	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the value of the Setup File Name setting for the current product configuration.
UpgradeCode	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the resulting setup package's Upgrade Code property.

Applies To

- ISWiProject

AddRelease Method

Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

- *InstallScript Object*
- *Merge Module*

Call the AddRelease method to add a release to the current product configuration. Using this method is similar to creating a release in the Release Wizard or the Releases view of the IDE.

Syntax

```
AddRelease (ReleaseKey As String) As ISWiRelease
```

Parameters

Table 12-80 • AddRelease Method Parameters

Parameter	Description
ReleaseKey	Pass the name of the release you would like to create in the current product configuration.

Applies To

- ISWiProductConfig

DeleteRelease Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the DeleteRelease method to delete a release from the current product configuration. Using this method is similar to deleting a release in the Releases view of the IDE.

Syntax

```
DeleteRelease (Release As ISWiRelease)
```

Parameters

Table 12-81 • DeleteRelease Method Parameters

Parameter	Description
ReleaseKey	Pass the name of the release you want to delete from the current product configuration.

Applies To

- ISWiProductConfig

ISWiProperty Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

ISWiProperty represents a Windows Installer property in the Property Manager of the InstallShield user interface. You can retrieve a property by specifying an item in the ISWiProperties collection.

Members

Table 12-82 • ISWiProperty Object

Name	Type	Description
Comments	Read-Write Property	Gets or sets the comments associated with the Windows Installer property.
Name	Read-Only Property	Contains the name of the property.
Value	Read-Write Property	Gets or sets the property's value.

Example

The following example sets the property ApplicationUsers to "OnlyCurrentUser," which means that the option "Only for me" is selected by default in the CustomerInformation dialog:

```
Dim pProject As ISWiProject

Set pProject = New ISWiProject
pProject.OpenProject "C:\MySetups\ISWiProperties.ism"

Dim pProperty As ISWiProperty
Set pProperty = pProject.ISWiProperties.Item("ApplicationUsers")
```

```
pProperty.Value = "OnlyCurrentUser"
pProperty.Comments = "Make ""Only for me"" the default radio button " & _
                     "in the CustomerInformation dialog."

pProject.SaveProject
```

Applies To

- ISWiProject

ISWiRelease Object



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Project-specific differences are noted where appropriate.

ISWiRelease represents an existing release in the Releases view of InstallShield. You can retrieve a release by specifying an item in the ISWiReleases collection. The release must have been created through the Release Wizard, the Releases view, or the command-line tool **ISCmdBld.exe**.

Members

Table 12-83 • ISWiRelease Object Members

Name	Project	Type	Description
BatchFileName	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Execute Batch File property, which specifies the filename of a batch file or executable file to execute after the release build is complete.
BrowsersToSupport	Basic MSI, InstallScript MSI	Read-Write Property	This property indicates which browsers a One-Click Install type of installation will support. Specify the following value: • **eocitbIE (0)**—Internet Explorer This property applies only to One-Click Web releases.
Build	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Method	Builds the current release. This is a full and complete build.
BuildErrorCount	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Only Property	After the Build method is run on the ISWiRelease object, BuildErrorCount contains the number of errors that occurred while the build process was running.
BuildLocation	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Gets or sets a string that identifies the top-level directory where the release will be built.
BuildTablesOnly	Basic MSI, InstallScript MSI, Merge Module	Method	Builds only the Windows Installer tables of the current release.
BuildTablesRefreshFiles	Basic MSI, InstallScript MSI	Method	Rebuilds your .msi file and updates the **Files** table, including any new or changed files in your release.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
BuildUTF8Database	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	If you want the Windows Installer database, along with any instance or language transforms, to be built using the UTF-8 encoding, set this Boolean property to True. The UTF-8 encoding supports characters from all languages simultaneously, enabling you to mix and match, for example, Japanese and German, or Russian and Polish, both in text shown to end users and in file names and registry keys. These mixed languages work correctly regardless of the current language of the target system. However, some scenarios result in user interface issues. For example, if an end user specifies the `/qb` command-line option or uninstalls the product from Add or Remove Programs, Windows Installer uses very small fonts to display the user interface text in a UTF-8 database. If you set this property to True, InstallShield creates an ANSI database when you build your release. This option does not let you mix characters from languages in different code pages.
BuildWarningCount	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Only Property	After the Build method is run on the ISWiRelease object, BuildWarningCount contains the number of warnings that occurred while the build process was running.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
CabCompressionType	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	If the Compressed property is set to True, you can use this CabCompressionType property to specify the type of compression that InstallShield should use when building this release's .cab files. Available options are: • **ecctLZX (3)**—InstallShield uses LZX compression to compress your product's data files into .cab files. This option results in the smallest .cab files; however, it also takes the most time to extract the data from these .cab files at run time. • **ecctMSZIP (1)**—InstallShield uses MSZIP compression to compress your product's data files into .cab files. • **ecctNone (0)**—InstallShield does not use any compression when creating the .cab files.  ***Important*** • *Using compression generally decreases the size of your compressed files, but the build process may take more time to complete. Depending on the number and size of the files being compressed, the LZX compression and the build may take hours to complete. Therefore, if you select the LZX option, it is recommended that your build machine have the latest hardware to minimize the time that it takes for the build to complete.*
CachePath	Basic MSI, InstallScript MSI	Read-Write Property	Specifies where cached installation files should be stored on the end user's system. Use a hard-coded value such as **C:\CachedFiles**. This property is used only if the CacheWebDownload property for the current build is set to True.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
CacheWebDownload	Basic MSI, InstallScript MSI	Read-Write Property	Specifies whether the installation files for the current build should be cached on the target system. Set this to True to cache the installation files on the target system for use with application maintenance and repair. Use the CachePath property to indicate where the cached files should be stored on the end user's system. This property is used only if the WebType property for the current build is set to ewtOneExe (2) (one executable).
CertificatePassword	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Write-Only Property	Sets the password for the .pfx file if the file has a password. InstallShield encrypts the password and stores it in your project file (.ism). At build time, InstallShield uses the password to sign files with a .pfx file. If your certificate is protected by a password but you do not enter it in this setting, signing with a .pfx file fails.  ***Note*** • *This property is not applicable if the certificate that you are using for signing is in a certificate store. Note that if the certificate was imported into the store with password protection, you will be prompted to enter the password at build time.*
CertificateURL	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Gets or sets the URL that is used in your digital certificate to link to a location that you would like end users to visit to learn more about your product, organization, or company. Use a fully qualified URL—for example, **http://www.mydomain.com**.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
Compressed	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Set this property to True if you want all of the package's files compressed into the .msi file, the .msm file, or Setup.exe, if selected. A value of False means that your application files are placed uncompressed in subfolders of the release location. Unlike the flexibility available in the Release Wizard's Custom Compression Settings panel, this property either compresses all or none of the files in the release.
CompressScript	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets the release's Compress Script property, which specifies whether the compiled script file (.inx file) is placed in a cabinet file (True) or is placed uncompressed in the Disk1 disk image folder (False).
CopyToFolder	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Copy To Folder property, which specifies a folder location to which to copy the release's disk image folders after the release build is complete. Existing folders with the same names as copied folders are overwritten but no folders are deleted.  ***Project*** • *For Basic MSI, InstallScript MSI, and Merge Module projects, use the DistributeLoc property instead of CopyToFolder to distribute your release to a folder.*
CreateAutorunINF	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to create an **Autorun.inf** file at the root of your release location.
CreatePDF	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to generate a package definition file for your installation, False otherwise.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
CubFile	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the name of the validation module (.cub file) that InstallShield uses to validate the built release.
DefaultLang	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Read-Write Property	Gets or sets the language identifier for the default language of a given release. This property value overrides the default project language that is configured in the General Information view or the String Editor view. For more information, see Setting the Default Project Language.
DelayMSIEngineReboot	Basic MSI, InstallScript MSI	Read-Write Property	Specifies whether you want to postpone any reboot associated with installing or updating the Windows Installer engine on the target system until after your installation has completed. Set this to True to postpone the reboot, if one is necessary. Set this to False to allow the system to reboot, if necessary, immediately after the Windows Installer engine has been installed or updated and before performing your installation. This option requires Windows Installer version 2.0.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
DigitalCertificateInfo	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Gets or sets the digital certificate that you want to use to sign your release. InstallShield lets you choose between the following options: • To specify the .pfx certificate file that you want to use for signing, set this property equal to the fully qualified path and file name of the .pfx file on your build machine. Instead of hard-coding a path, you can use a path variable that is defined in the Path Variables view or part of the ISWiPathVariables collection. At build time, InstallShield replaces the path variable with the appropriate value. • To reference a certificate store that contains the certificate that you want to use for signing, set this property equal to a certificate store string. To pass a certificate store string as the value, use the following convention: `*Store*StoreLocation:Cert Subject` For **Store**, pass one of the following values: • MY • CA • Trust • Root For **StoreLocation**, pass one of the following values: • User • Machine For **Cert Store**, pass the subject of the certificate that you want to use. To learn more, see Digital Signing and Security.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
DisplayDotNetOptionDialog	Basic MSI, DIM, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Set this property to True if you want the installation to display the .NET option message box at run time. This message box allows the end user to specify whether to install the .NET Framework. Set this property to False if the installation should not allow end users to determine whether they want to install the .NET Framework.  ***Note*** • *This property does not determine whether your installation includes the .NET Framework, only whether the end user has a choice to install it.*
DistributeAfterBuild	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Set this to True if you want the build engine to automatically distribute the current release after each build to the location that you specify for the DistributeLoc or DistributeToURLLoc properties.  ***Note*** • *When both the DistributeLoc and DistributeToURLLoc properties are specified, the release is copied to only the FTP location.*

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
DistributeLoc	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	If you want to be able to automatically distribute your release to a folder, specify the location for this property. The DistributeAfterBuild property must be set to True. Note that existing folders with the same names as copied folders are overwritten, but no folders are deleted. If the media format of the selected release is a network image, which creates only one disk image folder, the contents of the disk image folder, rather than the folder itself, are copied. If you chose to create a self-extracting executable file, the executable file, rather than the disk image folders, is copied.  ***Note*** • *When both the DistributeLoc and DistributeToURLLoc properties are specified, the release is copied to only the FTP location.*

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
DistributeToURLLoc	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	If you want to be able to automatically distribute your release to an FTP server, specify the FTP URL for the location for this property. The DistributeAfterBuild property must be set to True.  ***Note*** • *If you need to distribute your release to a path outside the FTP default folder, use a double slash (//). For example, to distribute your release to a root-level folder called* ***myproduct****, where the URL of the FTP server is* ***ftp://ftp.mydomain.com****, enter* **ftp://ftp.mydomain.com//myproduct** *for the FTP location.* *When both the DistributeLoc and DistributeToURLLoc properties are specified, the release is copied to only the FTP location.*
DistributeToURLPassword	Basic MSI, InstallScript MSI, Merge Module	Write-Only Property	If a password is required to upload to the FTP location that you are specifying, specify the password for this property.
DistributeToURLUserName	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	If a user name is required to upload to the FTP location that you are specifying, specify the user name for this property.
DotNetBaseLanguage	Basic MSI, InstallScript MSI	Read-Write Property	Set the language of the user interface of the .NET framework installation. If you set DotNetVersion to '1', then you must set this property.
DotNetBuildConfiguration	Basic MSI, InstallScript MSI	Read-Write Property	Set this to the active release name in your corresponding .NET Project file. Typical values would be *Release* and *Debug*.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
DotNetDelayReboot	Basic MSI, InstallScript MSI	Read-Write Property	Specifies whether you want to postpone any reboot associated with installing or updating the .NET Framework on the target system until after your installation has completed. Set this property to True to delay any prompts to reboot until after the installation is finished.  ***Note*** • *The .NET Framework may not function until after the target system is restarted. Therefore, it is strongly recommended that you set this property to False if the .NET Framework is used during the installation.*
DotNetFrameworkLocation	Basic MSI, InstallScript MSI	Read-Write Property	Set this to one of the following values to indicate how you want to package the .NET Framework redistributable into your installation: • **eelSourceMedia (0)**—Leaves the .NET Framework run time on the root of the source media. • **eelSetupExe (1)**—Extracts the .NET Framework from Setup.exe at run time. • **eelWeb (2)**—Downloads the .NET Framework run time from a default URL or a URL specified with the DotNetFrameworkURL property, described below. • **eelDotNetNone (3)**—Does not include the .NET Framework in the setup.
DotNetFrameworkURL	Basic MSI, InstallScript MSI	Read-Write Property	If DotNetFrameworkLocation is set to eelWeb (2), this property allows you to override the default location of dotnetfx.exe on the Internet.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
DotNetFxCmdLine	Basic MSI, InstallScript MSI	Read-Write Property	Set the command line that you want to pass to the setup inside of the .NET 1.1 redistributable (dotnetfx.exe).
DotNetLanguagePackCmdLine	Basic MSI, InstallScript MSI	Read-Write Property	Set the command line that you want to pass to the setup inside the .NET 1.1 language pack redistributables (langpack.exe).
DotNetLanguagePacks	Basic MSI, InstallScript MSI	Read-Write Property	Specify the language identifiers representing the additional .NET 1.1 language packs that you want to install.
DotNetUI	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True if you want to the user interface (UI) from the .NET Framework installation displayed at run time.
DotNetVersion	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to 1 if you want to ship .NET 1.1.
EnableDifference	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets the release's Differential Media property, which specifies whether the current release (the release that is selected in the Releases view) is a differential release—that is, a release that contains only those files that were absent from one or more of a specified set of existing releases—or a full release, which contains all your product's files, so that your product can be installed on a system on which no version of your product is currently installed.
EnableLangDlg	All	Read-Write Property	Set this property to True to display the Language Dialog from Setup.exe.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description

ExpirationDate	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets an expiration date (ISWiDate) for **Setup.exe**. If this property is set, end users cannot launch the **Setup.exe** file on or after the specified date.
			
			Following is sample Visual Basic code that sets the expiration date to December 30, 2012.
			
			`ISWiDate date`
			`date.wYear = 2012`
			`date.wMonth = 12`
			`date.wDay = 30`
			`pISWiRelease.ExpirationDate = date`
			
			If you use this property to get or set an expiration date, you can use the ExpirationMessage property to get or set the message that you want to be displayed at run time when an end user tries to launch **Setup.exe** on or after the expiration date.

ExpirationMessage	Basic MSI, InstallScript MSI	Read-Write Property	If you use the ExpirationDate property to set an expiration date, you can use the ExpirationMessage property to get or set the message that you want to be displayed at run time when an end user tries to launch **Setup.exe** on or after the expiration date. Following is sample Visual Basic code that sets an expiration message. `pISWiRelease.ExpirationMessage = "This setup expired on %s. The setup will now exit."`
FilterLangs	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to filter language-specific components according to your settings in the SupportedLangsData property, below.
FTPFolder	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's FTP Site Folder property, which specifies the folder on the FTP server to which you want to copy the disk image folders. This property is applicable only if the FTPHostAddress property is not empty.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
FTPHostAddress	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's FTP Host Address property, which specifies the Uniform Resource Locator (URL) of an FTP server to which the build process should upload the disk image folders. Leave this property empty if you do not want this release uploaded to an FTP server.
FTPPassword	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's FTP Site Password property, which specifies the password for the user name that you enter in the FTPUserName property. This property is applicable only if the FTPHostAddress property is not empty.
FTPUserName	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's FTP Site User Name property, which specifies the user name with which you want to gain access to the FTP server. This property is applicable only if the FTPHostAddress property is not empty.
GenerateFileHashValues	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Indicates whether you want to populate the MsiFileHash table for every unversioned file in your build. Set this to True to populate the MsiFileHash table for every unversioned file in your build. Set this to False if you do not want to populate the MsiFileHash table. If this property is set to False, InstallShield builds any entries that are found in the project's MsiFileHash table (populated using the Direct Editor). If you have already populated the MsiFileHash table for a particular file, the build uses that information instead of generating the information at build time.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
GenerateOneClickInstall	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to create a One-Click Install. If this property is set to True, you must specify file names in the OneClickHTMLBaseName and OneClickCabJarBaseName properties for this release. This property is ignored unless the release's Media Format property is set to Web via the Release Wizard's Media Type panel.
IFTWCabSizeInKb	Basic MSI, InstallScript MSI	Read-Write Property	Specifies the size, in kilobytes, of the cabinet (.cab) files built for the Web media type. Specify the value 0 (zero) to build a separate cabinet file for each component. This property is used only if the WebType property for the current release is set to ewtIFTW (1) (Install From The Web).
InitDlgProductName	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Init Dialog Product Name property, which specifies the product name to display in the setup initialization dialog box. If you leave this property empty, the product name that you specify in the General Information view.
JSharpCmdLine	Basic MSI, InstallScript MSI	Read-Write Property	Specify a command-line parameter to pass to vjredist.exe. Consult Microsoft Support for valid command-line parameters.
JSharpOptionalIfSilent	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to install the J# redistributable even if the J# Option dialog cannot be displayed (for example, if the installation is run silently).
JSharpOptionDlg	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to display the J# Option dialog, which asks whether the end user wants to install the J# redistributable. If .NET 1.1 is also included in the setup, the dialog states that .NET 1.1 will also be installed.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
JSharpRedistLocation	Basic MSI, InstallScript MSI	Read-Write Property	Set this to one of the following values to indicate how you want to package the J# redistributable into your setup. • **eelSourceMedia (0)**—Leaves the J# redistributable on the root of the source media. • **eelSetupExe (1)**—Extracts the J# redistributable from Setup.exe at run time. • **eelWeb (2)**—Downloads the J# redistributable from a default URL or a URL specified with the DotNetFrameworkURL property. • **eelDotNetNone (3)**—Does not include the J# redistributable in the setup.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
KeepUnusedDirectories	Basic MSI, InstallScript MSI, Merge Module	Read-Write Property	Specify whether you want InstallShield to remove unused directories from the **Directory** table of the .msi file when you build this release. Available options are: • **False**—If a directory that is listed in the Directory column of the **Directory** table is not referenced in any known location in the .msi file, InstallShield removes it from the **Directory** table of the .msi file that it creates at build time. For Basic MSI and InstallScript MSI projects, this occurs after any merge modules are merged, but only directories that are present in the .msi file are removed; therefore, if a merge module contains new unused directories in its **Directory** table, the new unused directories are added to the installation. • **True**—InstallShield does not remove any directories from the **Directory** table of the .msi file that it creates at build time. The default value is False.  ***Note*** • *Under some conditions, predefined directories cannot be resolved, causing an installation to fail. Removing unused directories from the* ***Directory*** *table enables you to avoid unnecessary failures. Therefore, it is recommended that you do not keep unused directories.*
LauncherCopyright	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Specifies the copyright information for the Properties dialog box of the **Setup.exe** file. You must set the UseMyVersionInfo property to True to override the default copyright information. To learn more, see Customizing File Properties for the Setup Launcher.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
LauncherFileDescription	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Specifies the file description for the Properties dialog box of the **Setup.exe** file. You must set the UseMyVersionInfo property to True to override the default file description. To learn more, see Customizing File Properties for the Setup Launcher.
LauncherPassword	Basic MSI, InstallScript MSI	Read-Write Property	Specifies a password to protect your application. You must set the PasswordProtectLauncher property to True to activate password protection. This property is applicable only to releases that meet the following criteria: • Single-file Setup.exe • Compressed files • Network image media type
MediaType	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Read-Only Property	Gets the type of distribution media for the release. Possible values are: • 0—CD-ROM • 1—Network image • 2—Custom • 3—DVD • 4—Web • 5—144 MB floppy disk • 6—InstallScript Object
MSI30EngineURL	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets the uniform resource locator (URL) for the location of the engine. This is the location that the installation file uses at run time to download the engine.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
MsiEngineLocation	Basic MSI, InstallScript MSI	Read-Write Property	Specifies where the MSI engine installers should be located. Set this property to one of the following: • **eelSourceMedia (0)**—Copy from the source media. Leaves the MSI engine installers you selected on the root of the source media. This option is not available for Web media types or Network Image media types with all of your files compressed into Setup.exe. • **eelSetupExe (1)**—Extract engine from Setup.exe. • **eelWeb (2)**—Download engine from the Web. If you use this setting, be sure to set the MSI30EngineURL property to appropriate Web locations.
Name	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Only Property	Name of the current release.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
ObjDiffOptions	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Object Difference property, which specifies the conditions for including InstallShield objects in your differential release. Use the following constants to set this property: • **eodoAll (0)**—The differential release will include all objects that would be included in the equivalent full update release. • **eodoExcludeAll (1)**—The differential release will not include any objects. • **eodoIncludeIfChanged (2)**—The differential release will include those objects that would be included in the equivalent full update release and that are not found in, or are different from, the corresponding object in at least one of the comparison releases. This property is applicable only if the EnableDifference property is set to True.
OneClickCabJarBaseName	Basic MSI, InstallScript MSI	Read-Write Property	Specifies the base file name for the cabinet (.cab) file generated by the build process for a One-Click installation. The ".cab" extension is appended to the name you specify here. The generated file (or files) will be created in the Disk1 folder for the current release. This property is used only if the GenerateOneClickInstall property for the current build is set to True.
OneClickHTMLBaseName	Basic MSI, InstallScript MSI	Read-Write Property	Specifies the base file name of the HTML file generated by the build process. The ".htm" extension is appended to the base name you specify, and the generated file is created in the Disk1 folder of the current release location. This property is used only if the GenerateOneClickInstall property for the current build is set to True.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
OptimizeSize	Basic MSI, InstallScript MSI	Read-Write Property	Specifies the level of compression to use in your release's cabinet files. Setting this property to True generally decreases the size of your compressed files, but the build process may take more time to complete. This property is used only if your release compresses some or all of its files.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
OSFilter	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Platform(s) setting, which specifies the operating systems that you want this release to support. If the platform specified for a component does not match one of the platforms that is selected for this property, the component is not included in the release. Use the following constants to set this property: ● eosOSIndepedent = 0 ● eosWin95 = &H10 (16) ● eosWin98 = &H40 (64) ● eosWinMe = &H80 (128) ● eosWinNT4 = &H10000 (65536) ● eosWin2000 = &H100000 (1048576) ● eosWinXP = &H400000 (4194304) ● eosWinServer2003 = &H800000 (8388608) ● eosWinVista = &H1000000 (16777216)—These constants are for Windows Vista and Windows Server 2008. ● eosWin7 = &H2000000 (33554432)—These constants are for Windows 7 and Windows Server 2008 R2. ● eosWin8 = &H4000000 (67108864)—These constants are for Windows 8 and Windows Server 2012. ● eosWin81 = &H8000000 (134217728)—These constants are for Windows 8.1 and Windows Server 2012 R2. ● eosAll = &HFD100D0 (265355472) You can specify multiple platforms; for example, **eosWin7 Or eosWinServer2008 Or eosWinVista Or eosWinServer2003**.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
PasswordProtectLauncher	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to password-protect your setup. Specify a password in the LauncherPassword property. This property is applicable only to releases that meet the following criteria: • Single-file Setup.exe • Compressed files • Network image media type
PathVariableOverrides	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Gets or sets the value of a path variable in your project for the specified release. If this property is used to set the value of a path variable, it overrides any value that is set in the Path Variables view. Note that you can use this variable to get or override user-defined path variables, environment variables, and registry variables that are configured in the Path Variables view; however, you cannot override predefined path variables such as *<WindowsFolder>*. Following is sample Visual Basic code that overrides the value of a path variable called PathVar1: `pISWiRelease.PathVariableOverrides = "PathVar1=C:\Test1" + vbCrLf + "PathVar2=C:\test2"` To learn more about path variables, see Using Path Variables.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
PostbuildEvent	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the command that you want to be run after InstallShield has built and signed the release. When you are specifying a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands. To learn more, see Specifying Commands that Run Before, During, and After Builds. Following is sample Visual Basic code that sets two commands: `pISWiRelease.PostbuildEvent = "Event1" + vbCrLf + "Event2"` At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one.  ***Edition*** • *This property is available in the Premier edition of InstallShield.*

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
PrebuildEvent	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the command that you want to be run before InstallShield starts building the release. This event runs after InstallShield creates the release folder and log file, but before InstallShield starts building the release. When you are specifying a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands. To learn more, see Specifying Commands that Run Before, During, and After Builds. Following is sample Visual Basic code that sets two commands: `pISWiRelease.PrebuildEvent = "Event1" + vbCrLf + "Event2"` At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one.  ***Edition*** • *This property is available in the Premier edition of InstallShield.*

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
PrecompressionEvent	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the command that you want to be run after InstallShield has built the .msi package and the .cab files (if your product's data files are to be stored in .cab files). Note that this event occurs after .cab files are streamed into the .msi package, but before the .msi package has been digitally signed and streamed into the **Setup.exe** file. When you are specifying a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands. To learn more, see Specifying Commands that Run Before, During, and After Builds. Following is sample Visual Basic code that sets two commands: `pISWiRelease.PrecompressionEvent = "Event1" + vbCrLf + "Event2"` At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one.  ***Edition*** • *This property is available in the Premier edition of InstallShield.*
PreProcessorDefines	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Compiler Preprocessor Defines setting, which specifies any preprocessor variable definitions.
PreviousPackage	Basic MSI, InstallScript MSI	Read-Write Property	Specifies the fully qualified path to a previous release (.msi file) to minimize the size of future patch packages.
ReleaseFlags	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets a string that contains the release flags that you want to use to filter your features. Separate multiple flags with a comma.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
RequiredExecutionLevel	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Gets or sets the minimum level required by your installation's **Setup.exe** file for running the installation (the setup launcher, any InstallShield prerequisites, and the .msi file) on Windows Vista and later platforms. The available options are: • **erelAsInvoker (0)**—**Setup.exe** does not require administrative privileges, and all users can run it without administrative privileges. **Setup.exe** does not display any UAC messages prompting for credentials or for consent. This is the default option for Basic MSI projects. • **erelAsHighestAvail (1)**—**Setup.exe** prefers administrative privileges. Administrators must authorize it; non-administrators run it without administrative privileges. This is the default option for InstallScript and InstallScript MSI projects. • **erelAsAdmin (2)**—**Setup.exe** requires administrative privileges to run. Administrators must authorize it; non-administrators must authenticate as an administrator. For more information, see Minimizing the Number of User Account Control Prompts During Installation.
SetupCmdLine	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Setup Command Line property, which specifies any command line parameters you want to pass to Setup.exe when the setup is launched.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
SetupEXE	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to create a **Setup.exe** setup launcher for the current release. The SetupEXE property and the TargetOS property correspond with the Setup Launcher setting on the Setup.exe tab in the Releases view. To learn about scenarios that require a setup launcher, see Creating a Setup Launcher.
ShallowFolderStructure	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to create the .msi file and related files directly in the Release Location without any of the subfolders.
ShowPasswordDlg	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets the release's Show Password Dialog property, which specifies whether to execute the password-checking code in the OnCheckMediaPassword event handler function's default code. This property is applicable only if you specify a non-null password in the Media Password property.
SignFiles	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	If you want to sign any of the files in your release, set this Boolean property to True and then use the SignFilesInclude and SignFilesExclude properties to indicate which files should be signed.  ***Windows Logo*** • *All executable files (including .exe, .dll, .ocx, .sys, .cpl, .drv, and .scr files) in an installation must be digitally signed for the Windows logo program.*

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
SignFilesExclude	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Specify any files and file patterns that you do not want to be digitally signed at build time. Note the following guidelines: • To indicate a wild-card character, use an asterisk (*). For example, if you do not want to sign any .drv files, specify the following: ***.drv** Using wild-card characters is especially helpful if you include dynamically linked files in your project and you want to avoid signing any files that match a certain pattern. • Put each file and each file pattern on its own line, with each separated by a carriage return. • Note that the files and file patterns that should not be signed override any files and file patterns that should be signed. For example, if you specify ***.exe** for the SignFilesInclude property and the SignFilesExclude property, InstallShield does not sign any .exe files.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
SignFilesInclude	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Specify the files and file patterns that you want to be digitally signed at build time. Note the following guidelines: • To indicate a wild-card character, use an asterisk (*). For example, if you want to sign all .exe files, specify the following: ***.exe** Using wild-card characters is especially helpful if you include dynamically linked files in your project and you want to sign all files that match a certain pattern. • Put each file and each file pattern on its own line, with each separated by a carriage return. • Note that the files and file patterns that should not be signed override any files and file patterns that should be signed. For example, if you specify ***.exe** for the SignFilesInclude property and the SignFilesExclude property, InstallShield does not sign any .exe files.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
SignFilesInPlace	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Specify whether you want InstallShield to sign your original files or just the files that are built into the release: • **False**—InstallShield signs a temporary copy of each file and then uses that signed temporary copy to build a release. Note that with this behavior, InstallShield will not modify or sign your original files. • **True**—If you want InstallShield to sign your original files, set this property to True. The default value of this property is set to False. The benefit of specifying True for a Basic MSI or InstallScript MSI project is that it helps create one patch that updates both compressed and uncompressed versions of a release that contains originally unsigned files.
SignLauncher	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to digitally sign your **Setup.exe** file.
SignMedia	This varies. (To learn more, see SignMedia.)	Read-Write Property	Set this property to indicate whether to digitally sign your media, and if so, which files should be signed.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
SignSignedFiles	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	If any of the files in your project are already digitally signed, specify whether you want InstallShield to replace those existing digital signatures with the digital signature that you specify through the DigitalCertificateInfo property: • **True**—Use the digital signature information that you are providing through the DigitalCertificateInfo property instead of any existing digital signature information that is already included with the file. • **False**—Leave the existing digital signature information intact for any files that are already signed. This is the default value. Note that this affects only files that meet the requirements that are specified in the SignFilesInclude and SignFilesExclude properties.
SingleEXEFileName	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Single Exe File Name setting, which specifies the file name (including extension) of a self-extracting executable file that runs the installation. If the property's value is null, no self-extracting executable file is created.
SingleEXEIconName	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Single Exe Icon File setting, which specifies the fully qualified name of the file from which the executable file's icon is taken at build time. If the property's value is null, a default icon is used.
SkinName	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Specify Skin setting, which specifies the dialog box skin that is applied to this release. This property is applicable only if the UseProjectSkin property is set to True.
SmallInitializationDialog	Basic MSI, InstallScript, InstallScript MSI	Read-Write Property	Set this property to display a small initialization dialog when the end user runs this release.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
SupportedLangsData	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets a list of language IDs that determines which components are included in the release. Only language-independent components and components of the specified languages are built into the release if the FilterLangs property is set to True. Use the decimal value of the language ID and separate multiple language with a comma.
SupportedLangsUI	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	Gets or sets a comma-delimited string of decimal language IDs that determines the release languages.
SupportedVersions	InstallScript, InstallScript Object	Read-Write Property	Gets or sets the release's Supported Version(s) property, which specifies a semicolon-delimited list of version numbers (for example, 1.2.3;1.2.4) of the earlier versions of your product to which this release can be applied as an update. If you leave this field blank, the setup can be run on a system on which any earlier version, or no version, of your product is currently installed. This property is applicable only if the EnableDifference property is set to False.
SuppressLauncherWarning	Basic MSI, InstallScript MSI	Read-Write Property	If the Windows Installer service cannot be installed or updated on a target system, the setup program displays a warning if this property is set to False. Set this to True to suppress the warning. This property applies only if you create a **Setup.exe** to launch your package.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
TargetOS	Basic MSI, InstallScript MSI	Read-Write Property	Gets or sets a value that determines which platform version of Windows Installer should be included in the release. Specify one of the following values: • **osNone (0)**—Do not include Windows Installer. • **osWin9x (1)**—Include only the Windows 9x version of **InstMsi.exe**. • **osWinNT (2)**—Include only the Windows NT version of **InstMsi.exe**. • **os9xNT (3)**—Include both the Windows NT and Windows 9x versions of **InstMsi.exe**. The SetupEXE property and the TargetOS property correspond with the Setup Launcher setting on the Setup.exe tab in the Releases view.  ***Note*** *• This setting applies to redistributables for Windows Installer 3.1 and earlier. For information on Windows Installer 4.5 redistributables, see Adding Windows Installer Redistributables to Projects. Note that Windows Installer 4.0 is not available as a redistributable.*
URLForYourFiles	Basic MSI, InstallScript MSI	Read-Write Property	Specifies the directory from where your data cabinet files will be downloaded. This property accepts a URL in the form **http://www.yourcompany.com/download**. This property is used only if the WebType property for the current release is set to ewtDownloader (0) (Downloader) or ewtIFTW (1) (Install from the Web).

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
UseMyVersionInfo	Basic MSI, InstallScript MSI	Read-Write Property	Set this property to True to override the default InstallShield copyright information for Setup.exe. You can specify your copyright information in the LauncherCopyright property. This property is applicable only to releases that meet the following criteria: • Single-file Setup.exe • Compressed files • Network image media type
UsePathVariableTestValues	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Read-Write Property	If you used test values for any of your path variables, set this property to True to set those variables to their actual values at this time.
UseProjectSkin	InstallScript, InstallScript Object	Read-Write Property	This Boolean property gets or sets whether the release's Specify Skin property is set to Use Project Setting, which uses the skin that is selected in the Dialogs view's Skins folder.
VMConfig	Basic MSI, InstallScript, InstallScript MSI	Read-Write String Property	 ***Edition*** • *This property is available in the Premier edition of InstallShield.* Gets or sets the name of the group of VM configuration settings that you want to use when distributing the selected release to a VM. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
VMMachinePassword	Basic MSI, InstallScript, InstallScript MSI	Read-Write String Property	***Edition*** • *This property is available in the Premier edition of InstallShield.* If a password is specified for the VM configuration that is selected through the VMConfig property, that password is considered to be the default value for the VMMachinePassword property. To override the password for the selected release, set the VMMachinePassword property. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.
VMMachineUserName	Basic MSI, InstallScript, InstallScript MSI	Read-Write String Property	***Edition*** • *This property is available in the Premier edition of InstallShield.* If a user name is specified for the VM configuration that is selected through the VMConfig property, that user name is considered to be the default value for the VMMachineUserName property. To override the user name for the selected release, set the VMMachineUserName property. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
VMSnapShot	Basic MSI, InstallScript, InstallScript MSI	Read-Write String Property	 ***Edition*** • *This property is available in the Premier edition of InstallShield.* Optionally gets or sets the name of the snapshot that you want to use for distributing the release. If a snapshot is specified for the VM configuration that is selected through the VMConfig property, the name of that snapshot is considered to be the default value for the VMSnapShot property. If no snapshot is specified for the VM configuration, InstallShield does not revert to a specific snapshot. InstallShield powers on the VM without reverting it to a specific snapshot, and copies your installation to the VM. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
VMStageMachineCopyPath	Basic MSI, InstallScript, InstallScript MSI	Read-Write String Property	***Edition*** • *This property is available in the Premier edition of InstallShield.* The copy path identifies the location on the VM where the release is to be distributed. The last subfolder in the path can be a path that InstallShield creates on the VM; the other folders in the path must already exist. If a destination path is specified for the VM configuration that is selected through the VMConfig property, that destination path is considered to be the default value for the VMStageMachineCopyPath property. To override the destination path for the selected release, set the VMStageMachineCopyPath property. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.
VMStagePostBuild	Basic MSI, InstallScript, InstallScript MSI	Read-Write Boolean Property	***Edition*** • *This property is available in the Premier edition of InstallShield.* Specify whether you want InstallShield to automatically distribute the selected release each time that it is successfully built. Distributing the release to the VM involves the following tasks: **1.** Revert the VM to the designated snapshot (if one is specified). **2.** Power on the VM. **3.** Copy the built release to the VM to make it available for testing. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
WebCreateDefaultPage	InstallScript	Read-Write Property	This Boolean property gets or sets the release's Create Default Web Page setting, which specifies whether InstallShield should create a default Web page (.htm file) and place it in the Disk1 folder.
WebPageUrl	InstallScript	Read-Write Property	Gets or sets the release's Web Page URL setting, which specifies the location of the Web page that should be launched if you click the Run From Web command on the Build menu in InstallShield: If nothing is entered for this property, InstallShield creates a **Setup.htm** file at build time and places it in the Disk1 folder. If you do enter a URL for this property, do not include the name of the page, and do not include an ending forward slash. For example, if the full URL for the **Setup.htm** file is **http://www.mypages.com/setup/Setup.htm**, enter the following as the Web page URL: **http://www.mypages.com/setup** When you click the Run from Web command on the Build menu, InstallShield launches the appropriate URL (**http://www.mypages.com/setup/Setup.htm**, in the aforementioned example).

Table 12-83 • ISWiRelease Object Members (cont.)

Name	Project	Type	Description
WebType	Basic MSI, InstallScript MSI	Read-Write Property	Indicates the configuration of your Web installation setup package. Set this property to one of the following: • **ewtDownloader (0)**—Downloader. Builds this release as a combination of Setup.exe and your MSI package, where the end user downloads and launches Setup.exe, which in turn downloads and runs the MSI database, which contains all of your files. • **ewtIFTW (1)**—Install from the Web. Builds this release as a combination of Setup.exe, your MSI database, and external cabinet (.cab) files. The end user downloads and runs Setup.exe, which in turn downloads and runs the MSI database. Based on the end user's setup type and feature selections, only the requested CAB files are downloaded and installed to minimize download time. Application maintenance and repair use the URL specified in the URLForYourFiles property as the installation source. • **ewtOneExe (2)**—One executable. Builds this release as a single self-extracting Setup.exe. This setting is ideal for a package that is to be downloaded from many Web or FTP sites, since the installation package is self-contained. Note that the entire installation package is downloaded, so the download time is greater than the ewtIFTW (1) (Install from the Web) setting.
WrapMSIIntoCab	Basic MSI, InstallScript MSI	Read-Write Property	Specifies whether to place the release's MSI database inside a cabinet (.cab) file. This property applies only if the WebType property for the current release is set to ewtDownloader (0) (Downloader).

Example

The following lines of Visual Basic demonstrate opening a project and showing a message box of several of a release's properties:

```
Dim pProject As ISWiProject

Set pProject = New ISWiProject
pProject.OpenProject "C:\MySetups\Commercial.ism", True

Dim pProdConfig As ISWiProductConfig
Set pProdConfig = pProject.ISWiProductConfigs.Item("Version 1")
' In an InstallScript project, there is only one element
' in ISWiProductConfigs, which is named "Media".

Dim pRelease As ISWiRelease
Set pRelease = pProdConfig.ISWiReleases.Item("NewRelease1")

Dim sProps As String
sProps = pRelease.Name & " has these properties:" & vbNewLine
sProps = sProps & "Build Location: " & pRelease.BuildLocation & vbNewLine
sProps = sProps & "Compressed: " & pRelease.Compressed & vbNewLine
sProps = sProps & "Default Language: " & pRelease.DefaultLang & vbNewLine
sProps = sProps & "Setup.exe: " & pRelease.SetupEXE


MsgBox sProps

pProject.CloseProject
```

Build Status Events

The ISWiRelease object raises the following status events:

- Public Event ProgressIncrement(ByVal lIncrement As Long, pbCancel As Boolean)
- Public Event ProgressMax(ByVal lMax As Long, pbCancel As Boolean)
- Public Event StatusMessage(ByVal sMessage As String, pbCancel As Boolean)

These events are raised during the build process and provide status updates. You can set pbCancel to stop the build. For sample code that demonstrates how to use these events, see Using Build Status Events.

Build Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Call the Build method to build the current release. This is a full and complete rebuild of your release.

Syntax

```
Build()
```

Applies To

- ISWiRelease

BuildTablesOnly Method



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

Call the BuildTablesOnly method to build only the Windows Installer tables of the current release.



Note *• If you have not built this setup already, a new .msi file is created, but no files are added to your setup. If you have built your setup already, the .msi file is updated when all the tables are built, but no files are transferred.*

Syntax

```
BuildTablesOnly ()
```

Applies To

- ISWiRelease

BuildTablesRefreshFiles Method



Project *• This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the BuildTablesOnly method to rebuild your MSI file and update the **Files** table, including any new or changed files in your installation.



Note *• Changed files are updated only if the size or time stamp differs from the copy already included in the build. References to deleted files are removed from the setup, but the file remains in the build location. This type of build can be run only after a complete build has been performed, and it works only when the media is an uncompressed network image.*

Syntax

```
BuildTablesRefreshFiles ()
```

Applies To

- ISWiRelease

CubFile Property



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*
- *Merge Module*

CubFile is a read-write property which you must specify prior to ISWiRelease.Build. This is a build option which is the same as building a release at the command line using **ISCmdBld.exe**.

The string parameter associated with this property is .cub file name.

Applies to

- ISWiRelease

SignMedia Property



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Project-specific differences are noted where appropriate.

The ISWiRelease.SignMedia property enables you to indicate whether to digitally sign your media, and if so, which files should be signed.

The following table lists the four states that can be assigned or retrieved from the ISWiRelease.SignMedia property.

Table 12-84 • ISWiRelease.SignMedia Property Values

Return Value	Type	Description
esBoth (3)	Read-Write Property	For Basic MSI and InstallScript MSI projects: Sign the Windows Installer package (.msi file) and **Setup.exe**. For InstallScript projects: Sign both the media header file (**Data1.hdr**) and **Setup.exe**.

Table 12-84 • ISWiRelease.SignMedia Property Values (cont.)

Return Value	Type	Description
esMsi (2)	Read-Write Property	For Basic MSI, InstallScript MSI, and Merge Module projects: Digitally sign only the Windows Installer package (.msi file or .msm file). For InstallScript projects: Sign the media header file (**Data1.hdr**).
esNone (4)	Read-Write Property	For Basic MSI, InstallScript, and InstallScript MSI projects: None of the media files (**Setup.exe**, Windows Installer package, or the header file) are signed at build time.
esSetupExe (1)	Read-Write Property	For Basic MSI, InstallScript, and InstallScript MSI projects: Digitally sign only **Setup.exe**.

ISWiRemoveFile Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

ISWiRemoveFile represents a file or folder removal entry in the InstallShield user interface. To retrieve a file or folder removal entry, specify an item in the ISWiRemoveFiles collection.

Members

Table 12-85 • ISWiRemoveFile Object Members

Name	Type	Description
FileKey	Read-Write Property	Gets or sets the FileKey property for this file or folder removal entry.
FileName	Read-Write Property	Gets or sets the FileName property for this file or folder removal entry.
DirProperty	Read-Write Property	Gets or sets the DirProperty property for this file or folder removal entry.
InstallMode	Read-Write Property	Gets or sets the InstallMode property for this file or folder removal entry. Specify one of the following types: • **erfimOnInstall (1)**—Remove the file or folder only when the associated component is being installed. • **erfimOnUninstall (2)**—Remove the file or folder only when the associated component is being removed. • **erfimOnBoth (3)**—Remove the file or folder when the associated component is being installed or removed.

ISWiSequenceRecord Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

ISWiSequenceRecord represents an action in the Custom Actions and Sequences view of the InstallShield user interface.

Members

Table 12-86 • ISWiSequenceRecord Object Members

Name	Type	Description
Name	Read-Write Property	Holds the string name of the specified action.
Comment	Read-Write Property	Gets or sets comments for the specified action.
Condition	Read-Write Property	Gets or sets the condition for the specified action.
Sequence	Read-Write Property	Gets or sets the sequence number for the specified action.

Example

The following VBScript code displays the sequence number of the LaunchConditions action in the project's Installation User Interface sequence.

```
Set oISM = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")

oISM.OpenProject "C:\MySetups\MyProject.ism", True ' open read-only

Set oSequenceRecord = oISM.InstallUISequence("LaunchConditions")

MsgBox "LaunchConditions is at sequence number " & oSequenceRecord.Sequence

oISM.CloseProject
```

Applies To

- ISWiSequence

ISWiSetupFile Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

ISWiSetupFile represents a support file in your project. The support file can be an existing one from the Support Files view (or the Support Files/Billboards view) in InstallShield or one that is added by calling AddSetupFile.

Use the project's ISWiSetupFiles collection to access a specific support file. You must specify either an index number or the file key for the Item property of ISWiSetupFiles.

Members

Table 12-87 • ISWiSetupFile Object Members

Name	Type	Description
FileName	Read-Only Property	Gets the name of the support file (without the path).
Language	Read-Write Property	Gets the string that identifies the language identifier for the support file; for example: `m_pFile.Language = "1033"`
Path	Read-Only Property	Gets the fully qualified path (including the file name) of the support file's source location. Any path variables are resolved to the actual path.
Splash	Read-Write Property	Specifies whether this support file is a splash screen (True) or not (False).
Target	Read-Write Property	Gets or sets the support file's target directory within **SUPPORTDIR**. The following line specifies the support file to be stored in the **MyFolder1\MyFolder2** folder under **SUPPORTDIR**: `m_pFile.Target = "MyFolder1\MyFolder2"`

Example

The file key is visible in the IDE only as the first entry in the Direct Editor view's **ISSetupFile** table. Instead of using the file key, you could write code similar to the following example to check for a certain file name:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pSetupFile As ISWiSetupFile

For Each pSetupFile In pProj.ISWiSetupFiles
    If UCase(pSetupFile.FileName) = "MYSUPPORTFILE.EXT" Then
        Exit For
    End If
Next

pProj.SaveProject
pProj.CloseProject
```

If the file name MySupportFile.ext is found among the project's setup files, the For loop ends, and pSetupFile holds a copy of the ISWiSetupFile object for MySupportFile.ext.

ISWiSetupType Object



Project • *This information applies to the following project types:*

- *InstallScript*

- *InstallScript MSI*

ISWiSetupType represents a setup type belonging to your project. The setup type can be an existing one from the Setup Types view in the InstallShield user interface or one added by calling AddSetupType.

Use the project's ISWiSetupTypes collection to access a specific setup type. You must specify either an index number or the setup type name for the Item property of ISWiSetupFiles. The setup type name is visible in the IDE in the Setup Types view's tree control.

Members

Table 12-88 • ISWiSetupType Object Members

Name	Type	Description
Name	Read-Only Property	Stores the name of the setup type.
Description	Read-Write Property	This string property specifies the setup type's description. (This description is displayed to end users during the installation process only if you call SdSetupTypeEx in your script.)
DisplayName	Read-Write Property	This string property specifies the name of this setup type as you would like it displayed to your users.

Example

The following Visual Basic code illustrates how you would set a property for a specific setup type:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pSetupType As ISWiSetupType

Set pSetupType = pProj.ISWiSetupTypes("CSSaveSpace")
pSetupType.DisplayName = "Client Setup - Save Space"

pProj.SaveProject
pProj.CloseProject
```

ISWiShellProperty Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The ISWiShellProperty object represents a shortcut property that you want the Windows Shell to set. The properties that the Shell can set are defined in **propkey.h**, which is part of the Windows SDK.

Members

Table 12-89 • ISWiShellProperty Object Members

Name	Type	Description
Name	Read-Write Property	Retrieves or sets the primary key name for a Shell property that is specified in the **MsiShortcutProperty** table.
PropertyName	Read-Write Property	Retrieves or sets a Shell property. The properties are defined in **propkey.h**, which is part of the Windows SDK. PropertyName typically consists of a GUID and a property ID; for example: **{9F4C2855-9F79-4B39-A8D0-E1D42DE1D5F3}, 9**
PropertyValue	Read-Write Property	Retrieves or sets a string value for the property.

ISWiShortcut Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

Project-specific differences are noted where appropriate.

The ISWiShortcut object represents a shortcut in the Shortcuts explorer. The shortcut can be an existing item in an ISWiShortcuts collection or one you just created by calling AddShortcut.

Use this object's properties to set the shortcut's attributes in much the same manner as you would in the IDE.

Members

Table 12-90 • ISWiShortcut Object Members

Name	Project	Type	Description
AddShellProperty	Basic MSI, InstallScript MSI, Merge Module	Method	Adds a Shell property for the current shortcut object.
Arguments	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets the command-line arguments that will be passed to the target file. You can substitute "%1" for the file name. The following statement might launch the target file in silent mode: `m_pShortcut.Arguments = "-s"`
DeleteShellProperty	Basic MSI, InstallScript MSI, Merge Module	Method	Deletes the specified Shell property from the current shortcut object.
Description	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Contains any descriptive text you have entered for this shortcut. The description is displayed as a tooltip. If this property uses a string entry, Description gives you the value for the string entry identified by the ActiveLanguage property of ISWiProject.
DisplayName	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Contains the display name you have entered for this shortcut. This name is a localizable value that will be used for the shortcut's name when it is installed instead of the name that identifies the shortcut in the Shortcuts explorer. If this property uses a string entry, DisplayName gives you the value for the string identified by the ActiveLanguage property of ISWiProject.

Table 12-90 • ISWiShortcut Object Members (cont.)

Name	Project	Type	Description
DoNotHighlightAsNew	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets a Boolean property that indicates whether the shortcut is configured to be highlighted on the Start menu as new after end users install the product. This has the same effect as clearing the **Highlight newly installed programs** check box in the Customize Start Menu dialog box for an individual item on a target system. To prevent the Start menu shortcut from being highlighted as newly installed, set this Boolean property to True. To enable the shortcut to be highlighted, set this Boolean property to False. For more information about this highlight functionality, see Preventing a Shortcut on the Start Menu from Being Highlighted as Newly Installed.
EnableWin8StartPin	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets a Boolean property that indicates whether the shortcut is configured to be pinned by default to the Start screen on Windows 8 target systems. To pin the shortcut by default, set this Boolean property to True. To disable pinning, set this Boolean property to False. For more information about this pinning functionality, see Suppressing Initial Pinning of a Shortcut to the Windows 8 Start Screen.
HotKey	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Stores the accelerator key for this shortcut as a Long.

Table 12-90 • ISWiShortcut Object Members (cont.)

Name	Project	Type	Description
IconFile	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Enter the fully qualified path to the file that contains the icon for this shortcut. You must specify either an executable file that contains the icon resource or an .ico file. (Windows Installer based projects only:) If you leave this property blank, the key file of the component will automatically be set as the shortcut's icon. Because Windows Installer requires a separate icon when the component is advertised, InstallShield extracts the icon from the executable you specify. You must set the IconIndex property if there is more than one icon resource in the executable.
IconIndex	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Contains the index for the specified icon as a Long if there is more than one icon resource in IconFile. A nonnegative integer refers to the order of the icon resources in the executable. For example, 0 is for the first icon in the file, 1 for the second, 2 for the third, and so on. Use a negative number to refer to a specific resource ID. For example, the icon index -12 points to the icon with a resource ID of 12.
InternetShortcut	InstallScript	Read-Write Property	Gets or sets the Internet Shortcut property for this shortcut, which specifies whether the shortcut is an Internet shortcut that points to the Uniform Resource Locator (URL) specified in the shortcut's Target property. This property is Boolean.
ISWiShellProperties	Basic MSI, DIM, InstallScript MSI, Merge Module	Collection	Contains all of the Shell properties for the current shortcut object.
Name	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Only Property	Retrieves the name of the shortcut. This is the same name that appears in the Shortcuts explorer; do not confuse it with the shortcut's Display Name property.

Table 12-90 • ISWiShortcut Object Members (cont.)

Name	Project	Type	Description
PreventPinning	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Gets or sets a Boolean property that indicates whether you want the context menu commands for pinning a shortcut to the taskbar and to the Start menu to be displayed after end users install your product. To hide the context menu commands for pinning the shortcut to the taskbar or Start menu, set this Boolean property to True. To display the context menu commands and allowing pinning, set this Boolean property to False. For more information about this pinning functionality, see Specifying Whether End Users Should Be Able to Pin a Shortcut to the Taskbar or Start Menu.
ReplaceExistingIfFound	InstallScript	Read-Write Property	Gets or sets the Replace Existing (If Found) setting for this shortcut, which specifies whether the shortcut should replace an already existing shortcut with the same display name in the same location on the target system. This property is Boolean.
ShowCmd	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Specify one of the following options for determining what mode the file is opened with: • **scNormal (1)**—The program launches in a normal-sized window. • **scMaximized (3)**—The program launches in full-screen view. • **scMinNoActive (7)**—The program launches in a minimized window, visible only on the taskbar.
Target	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Provide the path and file name for this shortcut's target file. Instead of hard-coding a path, you can use a Windows Installer directory property in square brackets—for example, [INSTALLDIR]MyApp.exe—or, in an InstallScript project, a system variable in angle brackets—<TARGETDIR>\MyApp.exe. (Windows Installer based projects only:) If you leave this property blank, the key file of this component will automatically be set as the shortcut's target.

Table 12-90 • ISWiShortcut Object Members (cont.)

Name	Project	Type	Description
Uninstall	InstallScript	Read-Write Property	Gets or sets the Uninstall property for this shortcut, which specifies whether the shortcut is removed when the application is uninstalled. This property is Boolean.
UserType	InstallScript	Read-Write Property	Gets or sets the Type property for this shortcut, which specifies whether you want the shortcut to appear in the list of personal shortcuts for the user installing your application, in the list of shortcuts that are common to all users, or if you would like the setup to decide automatically based on the value of the ALLUSERS system variable. Specify one of the following options for determining what mode the file is opened with: • **estAutomatic (0)**—The shortcut appears in the list of common shortcuts if ALLUSERS is non-zero, or in the list of personal shortcuts if ALLUSERS is FALSE. • **estCommon (1)**—The shortcut appears in the list of shortcuts that are common to all users. • **estPersonal (2)**—The shortcut appears in the list of personal shortcuts for the user installing your application.
WorkingDirectory	Basic MSI, InstallScript, InstallScript MSI, Merge Module	Read-Write Property	Specify the string that becomes the shortcut's "start in" folder. Instead of hard-coding a path, you can use a Windows Installer directory property in square brackets—for example, [INSTALLDIR]MyApp.exe—or, in an InstallScript project, a system variable in angle brackets—<TARGETDIR>\MyApp.exe.

Applies To

- ISWiFolder

AddShellProperty Method

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*

- *Merge Module*

The AddShellProperty method adds a Shell property for the current shortcut object. The properties that the Shell can set are defined in **propkey.h**, which is part of the Windows SDK.

For more information on setting Shell properties for a shortcut, see Setting Custom Shell Properties.

Syntax

```
Function AddShellProperty (sName As String) As ISWiShellProperty
```

Parameters

Table 12-91 • AddShellProperty Method Parameters

Parameter	Description
sName	Pass a string to specify the name of a Shell property that you want to add to the shortcut.

Return Values

The AddShellProperty method returns an ISWiShellProperty object.

Example

The following sample VBScript snippet adds a Shell property to a shortcut:

```
Dim pShellProperty As ISWiShellProperty
Set pShellProperty = pShortcut.AddShellProperty("MyShellProperty")
```

Applies To

- ISWiShortcut

DeleteShellProperty Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The DeleteShellProperty method deletes the specified Shell property from the current shortcut object.

For more information on setting Shell properties for a shortcut, see Setting Custom Shell Properties.

Syntax

```
DeleteShellProperty (pShellProp As ISWiShellProperty) As Long
```

Parameters

Table 12-92 • DeleteShellProperty Method Parameters

Parameter	Description
pShellProp	Pass the ISWiShellProperty object to specify a Shell property that you want to remove from a shortcut.

Example

The following sample VBScript snippet deletes all of the Shell properties from a shortcut:

```
Dim pShellProperty As ISWiShellProperty
    For Each pShellProperty In pShortcut.ISWiShellProperties
       pShortcut.DeleteShellProperty pShellProperty
    Next
```

Applies To

- ISWiShortcut

ISWiSISProperty Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

ISWiSISProperty represents a property in the Summary Information Stream. You can retrieve a property by specifying an item in the ISWiSISProperties collection. The following Summary Information Stream property names are supported:

- Title
- Subject
- Author
- Keywords
- PackageCode
- TemplateSummary
- Comments
- Schema

Several of these properties use string entries, as required by InstallShield. In that case, they contain the value for the string identified by the ActiveLanguage property of ISWiProject.

Members

Table 12-93 • ISWiSISProperty Object Members

Name	Type	Description
Name	Read-Only Property	Contains the property's name. Must be one of the existing Summary Information Stream properties listed above.
Value	Read-Write Property	Gets or sets the property's value. If this property uses a string entry, Value gives you the value for the string identified by the ActiveLanguage property of ISWiProject.

Example

The following VBScript example sets the property Title to **Windows Installer Setup Package**:

```
Option Explicit

Dim pProject
Set pProject = CreateObject ("IswiAutoAutomation Interface Version.ISWiProject")

pProject.OpenProject "C:\MySetups\SampleProject.ism"

Dim pTitle, sTitle
sTitle = "Windows Installer Setup Package"

Set pTitle = pProject.ISWiSISProperties.Item("Title")
pTitle.Value = sTitle

pProject.SaveProject
pProject.CloseProject
```

Applies To

- ISWiProject

ISWiSQLConnection Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

ISWiSQLConnection represents a SQL Server connection in the SQL Scripts view of the InstallShield interface. You can retrieve an ISWiSQLConnection object by using the ISWiSQLConnections collection.

Members

Table 12-94 • ISWiSQLConnection Object Members

Name	Type	Description
AddSQLRequirement	Method	Call the AddSQLRequirement method to target specific server versions and determine whether the SQL Server meets your application's minimum requirements.
AddSQLScript	Method	Call the AddSQLScript method to add a new script file to a SQL connection. ***Note*** • *The AddSQLScript method generates the ISSQLScriptFile table key with a name such as SQLScript1, SQLScript2, or SQLScript3. If you have multiple SQL script files in your project, it may be difficult to distinguish one from the other with this naming convention. Therefore, if you want to be able to specify the key, use the AddSQLScriptEx method.*
AddSQLScriptEx	Method	Call the AddSQLScriptEx method to add a new script file to a SQL connection, and generate a valid name from the passed string.
Authentication	Read-Write Property	This string gets or sets the authentication mode when connecting to a SQL Server.
Comments	Read-Write Property	This string property gets or sets any internal comments you want to include within your installation project.
Database	Read-Write Property	This string gets or sets the database name. At runtime, the installation uses this as the default database for installing the SQL script files contained in this connection. This value can be a property. Using a property allows you to dynamically set the name of this database at runtime.
DeleteSQLScript	Method	Call the DeleteSQLScript method to remove a SQL script file from a SQL Connection in the InstallShield interface.
GetDatabaseServer	Method	The GetDatabaseServer method returns an ISWiSQLDatabaseServer object.
InsertSQLScript	Method	Call the InsertSQLScript method to insert an existing script file to a SQL Connection in the InstallShield interface.
Name	Read-Write Property	This string property gets or sets the connection name. Type an internal name used to uniquely identify this connection in the setup project.

Table 12-94 • ISWiSQLConnection Object Members (cont.)

Name	Type	Description
Order	Read-Write Property	This integer property gets or sets the connection order.
Password	Read-Write Property	This string gets or sets the password when you choose to connect to a server using SQL Server authentication.
RemoveSQLRequirement	Method	Removes a specified SQL requirement from the target product.
SelectDatabaseServer(Name As String) As Boolean	Read-Write Property	This string gets or sets the selected database server name. At runtime, the installation will establish a connection and perform specified requirement checks on the designated database servers based on the sequence of the servers as they are listed.
Server	Read-Write Property	This string gets or sets the server name. At runtime, the installation uses this server name to connect to a SQL Server. This server will be used to install the SQL scripts contained within this connection. This value can be a property. Using a property allows you to dynamically set the name of this server at runtime. You must use the [Property] notation.
UserName	Read-Write Property	This string gets or sets the user name when you choose to connect to a server using SQL Server authentication.

Applies To

- ISWiProject

AddSQLRequirement Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The AddSQLRequirement adds a SQL requirement to the target product and returns an ISWiSQLRequirement object.

Syntax

```
Function AddSQLRequirement() As ISWiSQLRequirement
```

Return Values

This method returns ISWiSQLRequirement object.

Applies To

- ISWiSQLConnection

AddSQLScript Method



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the AddSQLScript method to add a new script file to a SQL connection.



Note *• The AddSQLScript method generates the ISSQLScriptFile table key with a name such as SQLScript1, SQLScript2, or SQLScript3. If you have multiple SQL script files in your project, it may be difficult to distinguish one from the other with this naming convention. Therefore, if you want to be able to specify the key, use the AddSQLScriptEx method.*

Syntax

```
Function AddSQLScript() As ISWiSQLScript
```

Return Values

This method returns an ISWiSQLScript object.

Applies To

- ISWiSQLConnection

AddSQLScriptEx Method



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the AddSQLScriptEx method to add an ISSQLScriptFile entry to a SQL connection. This method generates a valid name from the passed string. The method ensures that the name of the entry that is being added to the ISSQLScriptFile table is unique and less than 47 characters in length.

Syntax

```
Function AddSQLScriptEx (ByVal sCandidateName As String) As ISWiSQLScript
```

Parameters

Table 12-95 • AddSQLScriptEx Method Parameters

Parameter	Description
sCandidateName	Specifies the name of the script that you want to add to the SQL connection. Note that file names may contain invalid characters or may be too long for the key. However, if you pass a file name in this parameter, the AddSQLScriptEx method creates an entry with a valid key that is based on the file name; the result is the same as if you had added the SQL script to a connection in the SQL Scripts view. If you specify an empty string (""), the AddSQLScriptEx method generates the ISSQLScriptFile table key with a name such as SQLScript1, SQLScript2, or SQLScript3—the same as if the AddSQLScript method was used to add the ISSQLScriptFile entry.

Return Values

This method returns an ISWiSQLScript object.

Applies To

- ISWiSQLConnection

DeleteSQLScript Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the DeleteSQLScript method to remove a SQL script file from a SQL Connection in the InstallShield interface.

Syntax

```
Function DeleteSQLScript(Script As ISWiSQLScript) As Long
```

Parameters

Table 12-96 • DeleteSQLScript Method Parameters

Parameter	Description
Script	Specifies the name of the script you want to remove from the SQL connection in the InstallShield interface.

Applies To

- ISWiSQLConnection

GetDatabaseServer Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the GetDatabaseServer method to return an ISWiSQLDatabaseServer object.

Syntax

```
Sub SetPredefinedRequirement(RHS As ISWiSQLServerVersion)
```

Return Values

This method returns the ISWiSQLDatabaseServer object.

Applies To

- ISWiSQLConnection

InsertSQLScript Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the InsertSQLScript method to insert an existing script file to a SQL Connection in the InstallShield interface.

Syntax

```
Function InsertSQLScript(strScriptPath As String) As ISWiSQLScript
```

Parameters

Table 12-97 • InsertSQLScript Method Parameters

Parameter	Description
strScriptPath	Specifies the path to the script file you want to insert under the SQL Connection in the InstallShield interface.

Return Values

This method returns ISWiSQLScript object.

Applies To

- ISWiSQLConnection
- ISWiSQLDatabaseServer

RemoveSQLRequirement Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The RemoveSQLRequirement method removes a specified SQL requirement from the target product.

Applies To

- ISWiSQLConnection

ISWiSQLDatabaseServer Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

ISWiSQLDatabaseServer represents a database server that you can choose to target by version requirements at run time. You can retrieve an ISWiSQLDatabaseServer object by using the ISWiSQLDatabaseServers collection.

Members

Table 12-98 • ISWiSQLDatabaseServer Object Members

Name	Type	Description
InsertSQLScript	Method	Call the InsertSQLScript method to insert an existing script file to a SQL Connection in the InstallShield interface.
ISWiSQLRequirements	Collection	Returns the collection of ISWiSQLRequirements associated with this connection.
ISWiSQLScripts	Collection	Returns the collection of ISWiSQLScripts associated with this connection.
Name	Read-Write Property	This string property gets or sets the connection name for the database server.
Order	Read-Write Property	This integer property gets or sets the connection order.
Password	Read-Write Property	This string gets or sets the password when you choose to connect to a server using SQL Server authentication.
SelectDatabaseServer(Name As String) As Boolean	Read-Write Property	This string gets or sets the selected database server name. At runtime, the installation will establish a connection and perform specified requirement checks on the designated database servers based on the sequence of the servers as they are listed.
Server	Read-Write Property	This string gets or sets the server name. At runtime, the installation uses this server name to connect to a SQL Server. This server will be used to install the SQL scripts contained within this connection. This value can be a property. Using a property allows you to dynamically set the name of this server at runtime. You must use the [Property] notation.
UserName	Read-Write Property	This string gets or sets the user name when you choose to connect to a server using SQL Server authentication.

Applies To

- ISWiSQLConnection

ISWiSQLReplace Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

ISWiSQLReplace allows you to get and set properties in the Text Replacement tab associated with a script file in the InstallShield interface. You can retrieve an ISWiSQLReplace object using the ISWiSQLReplaces collection.

Members

Table 12-99 • ISWiSQLReplace Object Members

Name	Type	Description
Name	Read-Write Property	This string property gets or sets the name of the SQL Script Replacement object.
Search	Read-Write Property	This string property gets or sets the text string you want to find.
Replace	Read-Write Property	This string property gets or sets the text string with which you want to replace the existing string during text replacement.
MatchWholeWord	Read-Write Property	This is a Boolean property. During installation, this property helps identify only whole-word matches.
MatchCase	Read-Write Property	This is a Boolean property. During installation, this property indicates that the case for words found during text replacement must match.
ReplaceOnce	Read-Write Property	This is a Boolean property. During installation, this property indicates that the string you want to search and replace with a different string should be replaced only once.
PreserveCase	Read-Write Property	This is a Boolean property. During installation, this property indicates that case should be preserved during text replacement.

Applies To

- ISWiSQLScript

ISWiSQLRequirement Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

ISWiSQLRequirement represents the requirements property tab for a new SQL connection in the InstallShield interface. You can retrieve the ISWiSQLRequirement object using the ISWiSQLRequirements collection.

Members

Table 12-100 • ISWiSQLRequirement Object Members

Name	Type	Description
AnyGreaterServicePack	Read-Write Property	This is a Boolean property. Gets or sets a limit on the ServicePackLevel requirement.
AnyGreaterVersion	Read-Write Property	This is a Boolean property. Gets or sets a limit on the MajorVersion requirement.
Disable	Read-Write Property	This Boolean property disables the SQL requirement.
MajorVersion	Read-Write Property	This string property gets or sets the MajorVersion requirement.
Name	Read-Write Property	This string property gets or sets the name of the SQL Requirement object, which is necessary for determining which SQL Server versions are supported for a connection.
ServicePackLevel	Read-Write Property	This string property gets or sets the ServicePackLevel requirement.
SetPredefinedRequirement	Method	Configures the requirement from a set of predefined values which the installation author determines.

Applies To

- ISWiSQLConnection

SetPredefinedRequirement Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the SetPredefinedRequirement method to specify requirements for adding a SQL connection to your installation project.

Syntax

```
Sub SetPredefinedRequirement(RHS As ISWiSQLServerVersion)
```

Constants

The following is a list of constants associated with the ISWiSQLServerVersion object type.

- essv2K
- essv2Ksp1
- essv2Ksp2
- essv2Ksp3
- essv65
- essv65sp1
- essv65sp2
- essv65sp3
- essv65sp4
- essv65sp5
- essv65sp5a
- essv65sp5aUpdate
- essv7
- essv7sp1
- essv7sp2
- essv7sp3
- essv7sp4

Return Values

This method returns the ISWiSQLRequirement object.

Applies To

- ISWiSQLRequirement

ISWiSQLScript Object



Project *• This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Project-specific differences are noted where appropriate.

ISWiSQLScript represents a SQL script file in the SQL Scripts explorer of the SQL Scripts view in InstallShield. You can retrieve an ISWiSQLScript object by using the ISWiSQLScripts collection.

Members

Table 12-101 • ISWiSQLScript Object Members

Name	Type	Description
AddSQLScriptError	Method	Creates and returns ISWiSQLScriptError objects.
AddSQLScriptReplacement	Method	Creates and returns ISWiSQLReplace objects.
Comments	Read-Write Property	This string property gets or sets any internal comments you want to include within your installation project. You can enter any notes regarding the script. These notes will be saved and used only within the installation project.
Component	Read-Write Property	This string property gets or sets the name of the component which is associated with the feature that is associated with the SQL script object.
Condition	Read-Write Property	 ***Project*** • *This property is available for the following project types:* • *Basic MSI* • *DIM* • *InstallScript MSI* This property specifies the condition that is evaluated at run time to determine whether the SQL script should be run during installation or uninstallation. If the condition evaluates to true, the script is run. This property corresponds with the condition that is entered in the Script Condition area of the Runtime tab for a SQL script in the SQL Scripts view.
DeleteSQLReplace	Method	Deletes an ISWiSQLReplace object.
DeleteSQLScriptError	Method	Deletes an ISWiSQLScriptError object.
ErrorHandling	Read-Write Property	Gets or sets one of the following values. • **esehAbort (2)**—On error, abort the installation. • **esehGotoNextScript (0)**—On error, go to the next script. • **esehGotoNextStatement (1)**—On error, go to the next statement.
FullPath	Read-Write Property	This string property gets or sets the path of the .sql file associated with the SQL script object you want to execute by the installation.

Table 12-101 • ISWiSQLScript Object Members (cont.)

Name	Type	Description
InstallText	Read-Write Property	This string property gets or sets the status text when a script is run during install. This status text appears in the status message, which is displayed to the end-user while the script is running.
ISWiSQLScriptErrors	Read-Only Property	Returns the collection of ISWiSQLScriptErrors that are associated with this SQL script.
Name	Read-Write Property	This string property gets or sets the name of the SQL script object that you want to execute by the installation.
Order	Read-Write Property	This integer property gets or sets the order of execution of the script file.
RunOnInstall	Read-Write Property	This Boolean property specifies whether to schedule the execution of the SQL script during installation.
RunOnLogon	Read-Write Property	This Boolean property specifies whether to run the selected SQL script during login. This property corresponds with the Run Script During Login check box on the Runtime tab for a SQL script in the SQL Scripts view.
RunOnRollback	Read-Write Property	This Boolean property specifies whether to schedule the execution of the SQL script during rollback. This property corresponds with the Run Script During Rollback check box on the Runtime tab for a SQL script in the SQL Scripts view.
RunOnUninstall	Read-Write Property	This Boolean property specifies whether to schedule the execution of the SQL script during uninstallation.
UninstallText	Read-Write Property	This string property gets or sets the status text when a script is run during uninstall. This status text appears in the status message, which is displayed to the end-user while the script is running.
Version	Read-Write Property	This property gets or sets the version number for the SQL script file. For more information, see Specifying a Version Number for a SQL Script File.

AddSQLScriptError Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*

- *InstallScript*
- *InstallScript MSI*

Call the AddSQLScriptError method to add a custom error handling entry to a SQL script.

Syntax

```
Function AddSQLScriptError()
```

Return Values

This method returns an ISWiSQLScriptError object.

Applies To

- ISWiSQLScript

AddSQLScriptReplacement Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the AddSQLScriptReplacement method to specify a string and replace it with a different string in a SQL script file.

Syntax

```
Function AddSQLScriptReplacement() As ISWiSQLReplace
```

Return Values

This method returns the ISWiSQLReplace object.

Applies To

- ISWiSQLScript

DeleteSQLReplace Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*

- *InstallScript MSI*

Call the DeleteSQLReplace method to remove a text replacement entry that you have added.

Syntax

```
Function DeleteSQLReplace(Replace As ISWiSQLReplace) As Long
```

Applies To

- ISWiSQLScript

DeleteSQLScriptError Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

Call the DeleteSQLScriptError method to remove a custom error handling entry that you have added.

Syntax

```
Function DeleteSQLScriptError(ByVal pScriptError As ISWiSQLScriptError)
```

Applies To

- ISWiSQLScript

ISWiSQLScriptError Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

ISWiSQLScriptError represents a custom error handling entry for a SQL script file in the SQL Scripts view in InstallShield. You can retrieve an ISWiSQLScriptError object by using the ISWiSQLScriptErrors collection.

Members

Table 12-102 • ISWiSQLScriptError Object Members

Name	Type	Description
ErrorHandling	Read-Write Property	Gets or sets one of the following values. • **esehAbort (2)**—On error, abort the installation. • **esehGotoNextScript (0)**—On error, go to the next script. • **esehGotoNextStatement (1)**—On error, go to the next statement.
ErrorNumber	Read-Write Property	Gets or sets a numeric value that indicates the SQL error number for which you want to customize error handling.

ISWiStringEntry Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The ISWiStringEntry object represents a string entry for a specific language.

Members

Table 12-103 • ISWiStringEntry Object Members

Name	Type	Description
Id	Read-Write Property	Gets or sets the language-independent identifier for the string. Each string identifier in a project is linked to one or more values.
Value	Read-Write Property	Gets or sets the run-time string.
Comment	Read-Write Property	Gets or sets an internal note about the string entry.

Applies To

- ISWiLanguage

ISWiUpgradeTableEntry Object



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

ISWiUpgradeTableEntry represents an upgrade item in the Upgrades view of InstallShield. You can retrieve an upgrade item by specifying an item in the ISWiUpgradeTableEntries collection.

Members

Table 12-104 • ISWiUpgradeTableEntry Object Members

Name	Type	Description
Delete	Method	Deletes the StaticUpgradeEntry from the project file.
DisplayName	Read-Write Property	Gets or sets the name of the upgrade entry. This is the internal name that is displayed for an upgrade item in the Upgrades view.
UpgradeCode	Read-Write Property	Gets or sets the upgrade table entry's Upgrade Code. The upgrade code is a GUID representing a related set of products. It is used in the Upgrade Table to search for related versions of the product that are already installed.
VersionMin	Read-Write Property	Specifies the lower bound of the range of product versions detected by FindRelatedProducts.
VersionMax	Read-Write Property	Specifies the upper boundary of the range of product versions detected by FindRelatedProducts.
Language	Read-Write Property	Specifies the set of languages detected by FindRelatedProducts. Enter a list of numeric language identifiers (LANGID) separated by commas.
Attributes	Read-Write Property	Contains the bit flags specifying attributes of the Upgrade table.
Remove	Read-Write Property	The formatted string entered for this property must evaluate to a comma-delimited list of feature names.
ActionProperty	Read-Write Property	When the FindRelatedProducts action detects a related product installed on the system, it appends the product code to the property specified in this field.

Applies To

- ISWiProject

Delete Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the Delete method to remove the StaticUpgradeEntry from the project file.

Syntax

```
Delete()
```

Applies To

- ISWiUpgradeTableEntry

Automation Collections



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

Each InstallShield automation collection is a set of objects. For example, the ISWiFeatures collection is a set of ISWiFeature objects in the current project. This section describes each of the InstallShield automation collections.

ISWiAdvancedFiles Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

ISWiSAdvancedFiles is a collection that contains every advanced file (as an ISWiAdvancedFile object) in your project. For example, the following Visual Basic code illustrates how you would set a property for a specific advanced file:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
```

```
pProj.ISWiAdvancedFiles(1).Disk = edtOther

pProj.SaveProject
pProj.CloseProject
```

Members

Table 12-105 • ISWiAdvancedFiles Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of advanced files in the ISWiAdvancedFiles collection.

Item	Read-Only Property	Provide the index number or name of the advanced file to retrieve the ISWiAdvancedFile object. For example, the following statements create references to the first advanced file in the collection and the advanced file named CSSaveSpace:
		`Set pSetupFile1 = pProj.ISWiAdvancedFiles.Item(1)`
		`Set pSetupFile2 = pProj.ISWiAdvancedFiles.Item("CSSaveSpace")`
		`Item is the default property for ISWiAdvancedFiles, which means that pProj.ISWiAdvancedFiles.Item("CSSaveSpace") is equivalent to pProj.ISWiAdvancedFiles("CSSaveSpace").`

Applies To

- ISWiProject

ISWiAutomaticUpgradeEntries Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

ISWiAutomaticUpgradeEntries returns a collection of ISWiAutomaticUpgradeEntry items as they are defined in the IDE.

Members

Table 12-106 • ISWiAutomaticUpgradeEntries Collection Members

Property	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiAutomaticUpgradeEntries collection.
Item	Read-Only Property	Provide the index number or upgrade code in order to retrieve the ISWiAutomaticUpgradeEntries items. Item is the default property for ISWiAutomaticUpgradeEntries, which means that m_ISWiProject.ISWiAutomaticUpgradeEntries.Item("12345678-1234-1234-1234-123456789012") is equivalent to m_ISWiProject.ISWiAutomaticUpgradeEntries("12345678-1234-1234-1234-123456789012").

Example

```
Dim m_ISWiProj As ISWiProject
Dim m_AutomaticUpgradeEntries As ISWiAutomaticUpgradeEntries

Set m_ISWiProj = New ISWiProject
m_ISWiProj.OpenProject "C:\mysetups\build 34.ism"

For Each m_AutomaticUpgradeEntries In m_ISWiProject.ISWiAutomaticUpgradeEntries
    m_AutomaticUpgradeEntries.Name = "Upgrade Msi Item1"
Next

m_ISWiProj.SaveProject
m_ISWiProj.CloseProject
```

The following line illustrates how to access a particular automatic upgrade entry:

```
Set m_AutomaticUpgradeEntries = ISWiProject.ISWiAutomaticUpgradeEntries ("Upgrade Msi Item1")
```

Applies To

- ISWiProject

ISWiComponents Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

- *InstallScript Object*
- *Merge Module*

Project-specific differences are noted where appropriate.

The ISWiComponents collection contains all of the components associated with a specified feature. ISWiComponents provides an interface for enumerating all of the components in a feature. When ISWiComponents is a member of ISWiProject, it contains all of the components in the project, not just those belonging to a specified feature.

Members

Table 12-107 • ISWiComponents Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiComponents collection.
Item	Read-Only Property	Provide the index number or name of the component in order to retrieve the ISWiComponent object. The name of the component is case-sensitive—for example, "Help_Files" and "Help_files" are two different components. Item is the default property for ISWiComponents, which means that m_ISWiFeature.ISWiComponents.Item("Help_Files") is equivalent to m_ISWiFeature.ISWiComponents("Help_Files").

Examples

The following Visual Basic code opens a project, loops through each component in the feature NewFeature1, sets the RemoteInstallation property for each component to run from source, and finally saves and closes the project.

```
Dim m_ISWiProj As ISWiProject
Dim m_Feature As ISWiFeature
Dim m_Comp As ISWiComponent

Set m_ISWiProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
m_ISWiProj.OpenProject "C:\mysetups\build 34.ism"

Set m_Feature = m_ISWiProj.ISWiFeatures("NewFeature1")

If Not m_Feature Is Nothing Then
  For Each m_Comp In m_Feature.ISWiComponents
      m_Comp.RemoteInstallation = rfsSource
  Next
EndIf

m_ISWiProj.SaveProject
m_ISWiProj.CloseProject
```

The following line illustrates how to access a particular component:

```
Set m_Comp = m_ISWiProj.ISWiFeatures("MyFeature").ISWiComponents("MyComponent")
```

Applies To

- ISWiFeature
- ISWiProject

ISWiComponentSubFolders Collection



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

ISWiComponentSubFolders is a collection that contains every component subfolder (as an ISWiComponentSubFolder object) in the current component or component subfolder. For example, the following Visual Basic code illustrates how you would apply a method to a specific component subfolder:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pComponent As ISWiComponent

Set pFeature = pProj.ISWiFeatures.Item("MyFeature")
Set pComponent = pFeature.ISWiComponents.Item("MyComp")
pComponent.ISWiComponentSubFolders.Item("MyCompSubFolder").AddFile("C:\My Files\MyFile.ext")

pProj.SaveProject
pProj.CloseProject
```

Members

Table 12-108 • ISWiComponentSubFolders Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of component subfolders in the ISWiComponentSubFolders collection.

Item	Read-Only Property	Provide the index number or name of the component subfolder to retrieve the ISWiComponentSubFolder object. For example, the following statements create references to the first component subfolder in the collection and the component subfolder named MyCompSubFolder:
		`Set pComponentSubFolder1 = pProj.ISWiComponentSubFolders.Item(1)`
		`Set pComponentSubFolder2 = pFeature.ISWiComponentSubFolders.Item("MyCompSubFolder")`
		Item is the default property for ISWiComponentSubFolders, which means that pFeature.ISWiComponentSubFolders.Item("MyCompSubFolder") is equivalent to pFeature.ISWiComponentSubFolders("MyCompSubFolder").

Applies To

- ISWiComponent
- ISWiComponentSubFolder

ISWiConditions Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The ISWiConditions collection contains all of the conditions associated with a specified feature. ISWiConditions provides an interface for enumerating all of the conditions in a feature.

Members

Table 12-109 • ISWiConditions Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiConditions collection.
Item	Read-Only Property	Provide the index number of the condition in order to retrieve the ISWiCondition object. Item is the default property for ISWiConditions, which means that m_ISWiFeature.ISWiConditions.Item(2) is equivalent to m_ISWiFeature.ISWiConditions(2).

Applies To

- ISWiFeature

ISWiCustomActions Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The ISWiCustomActions collection contains all of the custom actions in the current project.

The VBScript example below displays a string with each custom action in the project:

```
Set pProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProject.OpenProject "C:\MySetups\MyProject.ism", True ' open read-only

sNames = "This project contains the following custom actions:" & vbNewLine

iCount = pProject.ISWiCustomActions.Count
For i = 1 to iCount
    sNames = sNames & vbTab & pProject.ISWiCustomActions(i).Name & vbNewLine
Next

pProject.CloseProject

MsgBox sNames
```

Members

Table 12-110 • ISWiCustomActions Collection Members

Name	Type	Description
Count	Read-Only Property	Returns the number of elements in the ISWiCustomActions collection.
Item	Read-Only Property	Provide the index number or name of the custom action in order to retrieve the ISWiCustomAction object. The name of the custom action is case-sensitive—for example, "MyAction" and "myAction" are two different actions. Item is the default property for ISWiCustomActions, which means that pProject.ISWiCustomActions.Item("Action1") is equivalent to pProject.ISWiCustomActions("Action1").

Applies To

- ISWiProject

ISWiDynamicFileLinkings Collection



Project • *This information applies to the following project types:*

Project-specific differences are noted where appropriate.

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

The ISWiDynamicFileLinkings collection contains all of the dynamic file links that are associated with a specified component.

Members

Table 12-111 • ISWiDynamicFileLinkings Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of dynamic file links in the ISWiDynamicFileLinkings collection.
Item	Read-Only Property	Provide the index number or name of the dynamically linked folder to retrieve the ISWiDynamicFileLinking object.

Applies To

- ISWiComponent

ISWiEnvironmentVars Collection



Project • *The ISWiEnvironmentVars collection applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The ISWiEnvironmentVars collection contains all of the environment variables that are associated with a specified component.

Members

Table 12-112 • ISWiEnvironmentVars Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiEnvironmentVars collection.
Item	Read-Only Property	Provide the index number or name of the environment variable to retrieve the ISWiEnvironmentVar object.

Applies To

- ISWiComponent

ISWiFeatures Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

ISWiFeatures is a collection that contains every feature (as an ISWiFeature object) in your project.

When ISWiFeatures is a member of ISWiProject, it contains all of the top-level features in the project. To retrieve a collection of subfeatures, access the Features property of the ISWiFeature object. The result is an ISWiFeatures collection that contains the immediate subfeatures of the current top-level feature. Repeat for further levels of subfeatures.

Members

Table 12-113 • Members of ISWiFeatures Collection

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiFeatures collection.

Item	Read-Only Property	Provide the index number or name of the feature to retrieve the ISWiFeature object. For example, the following statements create a copy of the first item in the collection and of the feature named Help_Files:
		`Set pFeat1 = m_ISWiProject.ISWiFeatures.Item(1)`
		`Set pFeat2 = m_ISWiProject.ISWiFeatures.Item("Help_Files")`
		The name of the feature is case-sensitive—for example, "Help_Files" and "Help_files" are two different features.
		Item is the default property for ISWiFeatures, which means that m_ISWiProject.ISWiFeatures.Item("Help_Files") is equivalent to m_ISWiProject.ISWiFeatures("Help_Files").

Example

The following Visual Basic code builds a string with all of the features and components in your project:

```
Dim pFeature As ISWiFeature
Dim pComponent As ISWiComponent

For Each pFeature In m_ISWiProject.ISWiFeatures
```

```
        ' Add feature to string
        strProject = strProject & "Feature: " & pFeature.Name & vbNewLine
        strProject = strProject & "Components: "
        For Each pComponent In pFeature.ISWiComponents
            ' Add feature's components to string
            strProject = strProject & pComponent.Name & " "
        Next
        strProject = strProject & vbNewLine
    Next
```

Applies To

- ISWiProject

ISWiFiles Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The ISWiFiles collection contains all of the files associated with a specified component or, in InstallScript projects, a component subfolder. ISWiFiles contains several read-write properties that allow you to set component properties and file attributes as you would in the IDE.

The following Visual Basic code illustrates how you would set a property for a specific file in a component. For the purposes of this example, assume that the variable m_oMyComp already has a reference to a component.

```
m_oMYComp.ISWiFiles("F1028_MyFile.dll").Hidden = True
```

Members

Table 12-114 • ISWiFiles Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiFiles collection.
Item	Read-Only Property	Provide the index number or file key of the file in order to retrieve the ISWiFile object. You cannot use the file name; you must specify the file key. You can view this value in the Direct Editor view (in the File column of the **File** table) or, in Windows Installer–based projects, the Key column of a component's file list. Be aware that file keys are subject to change when the files are dynamically linked. Item is the default property for ISWiComponents, which means that m_ISWiComponent.ISWiFiles.Item("F1036_MyFile.exe") is equivalent to m_ISWiComponent.ISWiFiles("F1036_MyFile.exe").

Applies To

- ISWiComponent

ISWiFolders Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

ISWiFolders contains every program folder belonging to a specified component.

At minimum, each component has the default folders listed below in its ISWiFolders collection. In addition, you could have added a predefined folder to the Shortcuts explorer by right-clicking on Shortcuts and choosing "Show Folder."

- [TaskBarFolder]
- [SendToFolder]

- [DesktopFolder]

The following code demonstrates accessing the ISWiFolder object that represents the taskbar:

```
Dim m_pFolder As ISWiFolder
Set m_pFolder = m_pComponent.ISWiFolders("[TaskBarFolder]")
```

The taskbar, in turn, has subfolders of its own. These subfolders are contained in its SubFolders property, which is itself an ISWiFolders collection. Continuing the example above, you can now create a copy of the object for the Start menu, which is a subfolder of the taskbar in the Shortcuts explorer:

```
Set m_pFolder = m_pFolder.SubFolders("[StartMenuFolder]")
```

We can go ever farther with this example and get a copy of the folder MyProgFolder under the Programs menu:

```
Set m_pFolder = m_pFolder.SubFolders("[ProgramMenuFolder]").SubFolders("MyProgFolder")
```

Members

Table 12-115 • ISWiFolders Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiFolders collection.
Item	Read-Only Property	Provide the index number or name of the folder in order to retrieve the ISWiFolder object. Item is the default property for ISWiFolders, which means that m_ISWiComponent.ISWiFolders.Item("[DesktopFolder]") is equivalent to m_ISWiComponent.ISWiFolders("[DesktopFolder]").

Applies To

- ISWiComponent

ISWiLanguages Collection

Edition • *Support for multilanguage installations is available with InstallShield Premier Edition.*

Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

- *Merge Module*

The ISWiLanguages collection contains all of the languages that are included in the current project.

Members

Table 12-116 • ISWiLanguages Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of languages in the ISWiLanguages collection.
Item	Read-Only Property	Provide the index number or name of the language to retrieve the ISWiLanguage object.

Applies To

- ISWiProject

ISWiObjects Collection



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

ISWiObjects is a collection that contains every InstallScript object (as an ISWiObject object) in your project. The following Visual Basic code illustrates the use of this collection:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"
Dim pFeature As ISWiFeature
Dim pObject As ISWiObject

Set pFeature = pProj.ISWiFeatures.Item(2)
Set pObject = pFeature.ISWiObjects.Item("@ismk2:755142B0-1EB4-11D3-8B09-00105A9846E9")
pFeature.RemoveObject pObject

pProj.SaveProject
pProj.CloseProject
```

Members

Table 12-117 • ISWiObjects Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of InstallScript objects in the ISWiObjects collection.

Item	Read-Only Property	Provide the index number or moniker of the InstallScript object to retrieve the ISWiObject object. (To get an InstallScript object's moniker, create a new Professional project in the IDE and add the InstallScript object to a feature, then go to the Direct Editor view's ISFeatureExtended table and see the Moniker column.) For example, the following statements create references to the first InstallScript object in the collection and the InstallScript object with moniker @ismk2:755142B0-1EB4-11D3-8B09-00105A9846E9:
		`Set pObject1 = pProj.ISWiObjects.Item(1)`
		`Set pObject2 = pFeature.ISWiObjects.Item("@ismk2:755142B0-1EB4-11D3-8B09-00105A9846E9")`
		Item is the default property for ISWiObjects, which means that pFeature.ISWiObjects.Item("@ismk2:755142B0-1EB4-11D3-8B09-00105A9846E9") is equivalent to pFeature.ISWiObjects("@ismk2:755142B0-1EB4-11D3-8B09-00105A9846E9").

Applies To

- ISWiFeature

ISWiPathVariables Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The ISWiPathVariables collection contains all of the path variables that are associated with the current project. You can add a path variable with the AddPathVariable method and delete a path variable with the DeletePathVariable method.

The following VBScript code retrieves a list of every path variable in the project and displays the path variables in a message box:

```
Option Explicit
  Dim pProject
  Set pProject = CreateObject ("ISWiAutoAutomation Interface Version.ISWiProject")

  pProject.OpenProject "C:\MySetups\ISWiPathVariables.ism", True

  Dim sPathVars
  Dim pPathVar

  For Each pPathVar In pProject.ISWiPathVariables
      sPathVars = sPathVars & pPathVar.Name & ": " & pPathVar.Value & vbNewLine
  Next

  WScript.Echo sPathVars
  pProject.CloseProject
```

The following line illustrates how to access a particular path variable:

```
Set pPathVar = pProject.ISWiPathVariables.Item("MyPathVar")
```

Important • *Predefined path variables cannot be modified or deleted. If you attempt to do so, exception error 3142 occurs. For more information, see Predefined Path Variables.*

Members

Table 12-118 • ISWiPathVariables Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of path variables in the ISWiPathVariables collection.
Item	Read-Only Property	Provide the index number or name of the path variable to retrieve the ISWiPathVariable object.

Applies To

- ISWiProject

ISWiProductConfigs Collection

Project • *This information applies to the following project types:*

- *Basic MSI*

- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

InstallScript and InstallScript Object projects contain only one element in ISWiProductConfigs; it is named Media.

The ISWiProductConfigs collection contains all of the product configurations in the current project.

Members

Table 12-119 • ISWiProductConfigs Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiProductConfigs collection.
Item	Read-Only Property	Provide the index number or name of the product configuration in order to retrieve the ISWiProductConfig object. The name of the product configurations is case-sensitive—"Product One" and "Product one" are two different configurations. Item is the default property for ISWiProductConfigs, which means that pProject.ISWiProductConfigs.Item("Version 1") is equivalent to pProject.ISWiProductConfigs("Version 1").

Example

The VBScript example below simply displays a string with each product configuration in the project:

```
Dim pProject
Set pProject = CreateObject ("IswiAutoAutomation Interface Version.ISWiProject")
pProject.OpenProject "C:\MySetups\Vegetarian.ism", True

Dim sNames
sNames = "This project contains the following product configurations: " & vbNewLine & vbTab

Dim pProductConfigs
For Each pProductConfigs In pProject.ISWiProductConfigs
    sNames = sNames & pProductConfigs.Name & vbNewLine & vbTab
Next

pProject.CloseProject

WScript.Echo sNames
```

The following line illustrates how to access a particular product configuration:

```
Set pProdConfig = pISWiProj.ISWiProductConfigs.Item("Version 1.1")
```

Applies To

- ISWiProject

ISWiProperties Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The ISWiProperties collection contains all of the Windows Installer properties in the current project. This interface is similar to the Property Manager. You can add a property with the AddProperty method and delete a property with the DeleteProperty method.

The VBScript code below retrieves a list of every property in the project and displays them in a message box:

```
Option Explicit

Dim pProject
Set pProject = CreateObject ("ISWiAutoAutomation Interface Version.ISWiProject")

pProject.OpenProject "C:\MySetups\ISWiProperties.ism", True

Dim sProps
Dim pProp

For Each pProp In pProject.ISWiProperties
    sProps = sProps & pProp.Name & ": " & pProp.Value & "  // " & _
             pProp.Comments & vbNewLine
Next

WScript.Echo sProps

pProject.CloseProject
```

Members

Table 12-120 • ISWiProperties Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiProperties collection.
Item	Read-Only Property	Provide the index number or name of the property in order to retrieve the ISWiProperty object. The name of the property is case-sensitive—"UpgradeCode" and "Upgradecode" are two different properties. Item is the default property for ISWiProperties, which means that pISWiProject.ISWiProperties.Item("UpgradeCode") is equivalent to pISWiProject.ISWiProperties("UpgradeCode").

Applies To

- ISWiProject

ISWiReleases Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The ISWiReleases collection contains all of the releases belonging to the current product configuration.

The following line illustrates how to access a particular ISWiRelease object:

```
Set pRelease = pProject.ISWiProductConfigs("Version 1").ISWiReleases("NewRelease1")
' In an InstallScript project, there is only one element
' in ISWiProductConfigs, which is named "Media".
```

Members

Table 12-121 • ISWiReleases Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiReleases collection.
Item	Read-Only Property	Provide the index number or name of the release in order to retrieve the ISWiRelease object. The name of the release is case-sensitive—"NewRelease1" and "Newrelease1" are two different releases. Item is the default property for ISWiReleases, which means that m_ISWiBldLbl.ISWiReleases.Item("NewRelease1") is equivalent to m_ISWiBldLbl.ISWiReleases("NewRelease1").

Applies To

- ISWiProductConfig

ISWiRemoveFiles Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

The ISWiRemoveFiles collection represents a collection of ISWiRemoveFile objects in the InstallShield user interface.

Members

Table 12-122 • ISWiRemoveFiles Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiRemoveFiles collection.
Item	Read-Only Property	Provide the index number or key of the file or folder removal entry in order to retrieve its ISWiRemoveFile object. You must specify the key. You can view this value in the Direct Editor view (in the FileKey column of the **RemoveFile** table. Item is the default property for ISWiRemoveFiles, which means that m_ISWiComponent.ISWiRemoveFiles.Item("RemoveFile1") is equivalent to m_ISWiComponent.ISWiRemoveFiles("RemoveFile1").

Applies To

- ISWiComponent

ISWiSequence Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The ISWiSequence collection is a virtual collection that contains all of the actions of the current project, sorted by sequence number.

Members

Table 12-123 • ISWiSequence Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSequence collection.
InsertCustomAction	Method	Use this method to insert a custom action into the selected sequence. This method returns an ISWiSequenceRecord object.
Item	Read-Only Property	Provide the index number or name of an action in order to retrieve its ISWiSequenceRecord object. The name of the action is case-sensitive. That is, "ThisAction" and "thisAction" are two different actions. Item is the default property for ISWiSequence, which means that pProject.InstallUISequence.Item("ThisAction") is equivalent to pProject.InstallUISequence("ThisAction").
RemoveSequenceRecord	Method	Use this method to remove an item from a sequence in the project.

Example

The VBScript example below displays a string with each action name and sequence number in the project's Installation User Interface sequence:

```
Dim pProject
Set pProject = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProject.OpenProject "C:\MySetups\YourProject.ism", True ' open read-only

Dim sNames
sNames = "The Installation UI sequence contains the following actions:" & vbNewLine & vbTab

For i = 1 to pProject.InstallUISequence.Count
    sNames = sNames & pProject.InstallUISequence.Item(i).Name & " " & _
             pProject.InstallExecuteSequence.Item(i).Sequence & vbNewLine & vbTab
Next

pProject.CloseProject

MsgBox sNames
```

Applies To

- ISWiProject

InsertCustomAction Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the InsertCustomAction method to insert a custom action into one of the project's sequences.

Syntax

```
InsertCustomAction (CustomAction As ISWiCustomAction, Comment As String, Condition as String,
SequenceNumber as Long) as ISWiSequenceRecord
```

Parameters

Table 12-124 • InsertCustomAction Method Parameters

Parameter	Description
CustomAction	The custom action you want to create.
Comment	An optional string comment about your custom action.
Condition	An optional string condition for your custom action.
SequenceNumber	The numeric sequence number for your custom action.

Return Values

The InsertCustomAction method returns an ISWiSequenceRecord object.

Applies To

- ISWiSequence

RemoveSequenceRecord Method



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

Call the RemoveSequenceRecord method to remove an item from a sequence in the current project.

Syntax

```
(sName As String) As Long
```

Parameters

Table 12-125 • RemoveSequenceRecord Method Parameters

Parameter	Description
sName	Pass an ISWiSequenceRecord object to specify a sequence record that you want to remove from the project.

Return Values

Table 12-126 • RemoveSequenceRecord Method Values

Return Value	Description
0	The item was successfully removed from a sequence.
1	The item was not removed from a sequence.

Applies To

- ISWiSequence

ISWiSetupFiles Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*

ISWiSetupFiles is a collection that contains every support file (as an ISWiSetupFile object) in your project.

Members

Table 12-127 • ISWiSetupFiles Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of support files in the ISWiSetupFiles collection.

Item	Read-Only Property	Provide the index number or name of the support file to retrieve the ISWiSetupFile object. For example, the following statements create references to the first support file in the collection and the support file named MySupportFile.bmp:
		`Set pSetupFile1 = pProj.ISWiSetupFiles.Item(1)`
		`Set pSetupFile2 = pProj.ISWiSetupFiles.Item("MySupportFile.bmp")`
		Item is the default property for ISWiSetupFiles, which means that pProj.ISWiSetupFiles.Item("MySupportFile.bmp") is equivalent to pProj.ISWiSetupFiles("MySupportFile.bmp").

Example

The following Visual Basic code illustrates how you would set a property for a specific support file:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"

pProj.ISWiSetupFiles(1).Splash = True

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

ISWiSetupTypes Collection



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

ISWiSetupTypes is a collection that contains every setup type (as an ISWiSetupType object) in your project.

Members

Table 12-128 • ISWiSetupTypes Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of setup types in the ISWiSetupTypes collection.

Item	Read-Only Property	Provide the index number or name of the setup type to retrieve the ISWiSetupType object. For example, the following statements create references to the first setup type in the collection and the setup type named CSSaveSpace:
		`Set pSetupFile1 = pProj.ISWiSetupTypes.Item(1)`
		`Set pSetupFile2 = pProj.ISWiSetupTypes.Item("CSSaveSpace")`
		Item is the default property for ISWiSetupTypes, which means that pProj.ISWiSetupTypes.Item("CSSaveSpace") is equivalent to pProj.ISWiSetupTypes("CSSaveSpace").

Example

The following Visual Basic code illustrates how you would set a property for a specific setup type:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("IswiAutoAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.ism"

pProj.ISWiSetupTypes("CSSaveSpace").DisplayName = "Client Setup - Save Space"

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

ISWiShellProperties Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

ISWiShellProperties is a collection of all of the Shell properties for a specified ISWiShortcut object.

Members

Table 12-129 • ISWiShellProperties Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiShellProperties collection.
Item	Read-Only Property	Provide the index number or name of the Shell property in order to retrieve the ISWiShellProperty object.

Example

The following sample VBScript snippet retrieves the ISWiShellProperty object for the primary key MyShellProperty from the ISWiShellProperties collection:

```
Dim pShellProperty As ISWiShellProperty
Set pShellProperty = pShortcut.ISWiShellProperties("MyShellProperty")
```

Applies To

- ISWiShortcut

ISWiShortcuts Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *Merge Module*

ISWiShortcuts contains every shortcut belonging to a specified ISWiFolder object.

The following example makes a copy of the shortcut named MyShortcut under the program folder contained in m_pFolder:

```
Dim m_pShortcut As ISWiShortcut
Set m_pShortcut = m_pFolder.ISWiShortcuts("MyShortcut")
```

Members

Table 12-130 • ISWiShortcuts Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiShortcuts collection.
Item	Read-Only Property	Provide the index number or name of the shortcut in order to retrieve the ISWiShortcut object. Item is the default property for ISWiShortcuts, which means that m_pFolder.ISWiShortcuts.Item("MyShortcut") is equivalent to m_pFolder.ISWiShortcuts("MyShortcut").

Applies To

- ISWiFolder

ISWiSISProperties Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript MSI*
- *Merge Module*

This collection contains all of the values for the Summary Information Stream settings in the current project. By accessing an element in this collection, you can set all of the Summary Information Stream settings in the General Information view.

The following Visual Basic code reads every Summary Information Stream setting in your project and displays each setting and its value.

```
Dim pProject As ISWiProject

Set pProject = New ISWiProject
pProject.OpenProject "C:\MySetups\SampleApp.ism", True

Dim sSISProps As String
sSISProps = "This package's Summary Information Stream has the following " & _
            pProject.ISWiSISProperties.Count & " properties:"

Dim pSISProp As ISWiSISProperty
For Each pSISProp In pProject.ISWiSISProperties
    sSISProps = sSISProps & vbNewLine & vbTab & _
                pSISProp.Name & ": " & pSISProp.Value
Next

MsgBox sSISProps
```

```
pProject.CloseProject
```

Members

Table 12-131 • ISWiSISProperties Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSISProperties collection. Since the Summary Information Stream contains seven items at present, Count always has a value of 7.
Item	Read-Only Property	Provide the index number or name of the property in order to retrieve the ISWiSISProperty object. The name of the property is case-sensitive—"TemplateSummary" and "Templatesummary" are two different properties. Item is the default property for ISWiSISProperties, which means that pISWiProject.ISWiSISProperties.Item("Subject") is equivalent to pISWiProject.ISWiSISProperties("Subject").

Applies To

- ISWiProject

ISWiSQLConnections Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The ISWiSQLConnections collection contains all of the SQL Server connections in the current project.

Members

Table 12-132 • ISWiSQLConnections Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSQLConnections collection.
Item	Read-Only Property	Provide the index number or name of the connection in order to retrieve the ISWiSQLConnection object. The name of the connection is case-sensitive—"SQLConnection" and "SQLconnection" are two different items. Item is the default property for ISWiSQLConnections, which means that pISWiProject.ISWiSQLConnections.Item("SQLConnection") is equivalent to pISWiProject.ISWiSQLConnections("SQLConnection").

Applies To

- ISWiProject

ISWiSQLDatabaseServers Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The ISWiSQLDatabaseServers collection contains all the database servers that you can choose to target by version requirements at run time.

Members

Table 12-133 • ISWiSQLDatabaseServers Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSQLDatabaseServers collection.
Item	Read-Only Property	Provide the index number or name of the property in order to retrieve the ISWiSQLDatabaseServer object. The name of the database server is case-sensitive—"SQLDatabaseServer" and "SQLDatabaseserver" are two different items. Item is the default property for ISWiSQLDatabaseServers, which means that pISWiSQLConnection.ISWiSQLDatabaseServers.Item("SQLDatabaseServer") is equivalent to pISWiSQLConnection.ISWiSQLDatabaseServers("SQLDatabaseServer").

Applies To

- ISWiSQLConnection

ISWiSQLReplaces Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The ISWiSQLReplaces collection contains all the ISWiSQLReplace objects in the current project.

Members

Table 12-134 • ISWiSQLReplaces Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSQLReplaces collection.
Item	Read-Only Property	Provide the index number or name of the property in order to retrieve the ISWiSQLReplace object. The name of the collection is case-sensitive—"SQLReplaces" and "SQLreplaces" are different. Item is the default property for ISWiSQLReplaces, which means that pISWiSQLScript.ISWiSQLReplaces.Item("SQLReplace") is equivalent to pISWiSQLScript.ISWiSQLReplaces("SQLReplace").

Applies To

- ISWiSQLScript

ISWiSQLRequirements Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

ISWiSQLRequirements returns a collection of all ISWiSQLRequirement objects associated with the database server.

Members

Table 12-135 • ISWiSQLRequirements Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSQLRequirements collection.
Item	Read-Only Property	Provide the index number or name of the property in order to retrieve the ISWiSQLRequirement object. The name of the property is case-sensitive—"SQLRequirements" and "SQLrequirements" are different. Item is the default property for ISWiSQLRequirements, which means that pISWiSQLConnection.ISWiSQLRequirements.Item("SQLRequirement") is equivalent to pISWiSQLConnection.ISWiSQLRequirements("SQLRequirement").

Applies To

- ISWiSQLConnection

ISWiSQLScriptErrors Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The ISWiSQLScriptErrors collection contains all of the custom SQL script error handling entries for the ISWiSQLScriptError object.

Members

Table 12-136 • ISWiSQLScriptErrors Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSQLScriptErrors collection.
Item	Read-Only Property	Provide the index number or name of the property in order to retrieve the ISWiSQLScriptError object.

Applies To

- ISWiSQLScript

ISWiSQLScripts Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*

The ISWiSQLScripts collection contains all the SQL scripts in the SQL Connection.

Members

Table 12-137 • ISWiSQLScripts Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSQLScripts collection.
Item	Read-Only Property	Provide the index number or name of the property in order to retrieve the ISWiSQLScript object. The name of the script is case-sensitive—"SQLScript" and "SQLscript" are two different items. Item is the default property for ISWiSQLScripts, which means that pISWiSQLConnection.ISWiSQLScripts.Item("SQLScript") is equivalent to pISWiSQLConnection.ISWiSQLScripts("SQLScript").

Applies To

- ISWiSQLConnection

ISWiStringEntries Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *DIM*
- *InstallScript*
- *InstallScript MSI*
- *InstallScript Object*
- *Merge Module*

The ISWiStringEntries collection contains all of the string entries in the current language.

Members

Table 12-138 • ISWiStringEntries Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of string entries in the ISWiStringEntries collection.
Item	Read-Only Property	Provide the index number or name of the string entry to retrieve the ISWiStringEntry object.

Applies To

- ISWiLanguage

ISWiUpgradeTableEntries Collection



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

The ISWiUpgradeTableEntries collection contains all the static major upgrade items, as they are defined in the Upgrades view as well as the **Upgrade** table in the Direct Editor.

Members

Table 12-139 • ISWiUpgradeTableEntries Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiUpgradeTableEntries collection.
Item	Read-Only Property	Provide the index number or upgrade code in order to retrieve the ISWiUpgradeTableEntries items. Item is the default property for ISWiUpgradeTableEntries, which means that m_ISWiProject.ISWiUpgradeTableEntries.Item("12345678-1234-1234-1234-123456789012") is equivalent to m_ISWiProject.ISWiUpgradeTableEntries("12345678-1234-1234-1234-123456789012").

Example

```
Dim oProj As ISWiProject
Dim oUpgradeItem As ISWiUpgradeTableEntry

Set oProj = New ISWiProject
oProj.OpenProject "C:\MySetup.ism"

For Each oUpgradeItem In oProj.ISWiUpgradeTableEntries
    oUpgradeItem.Language = "1033"
Next

oProj.SaveProject
oProj.CloseProject
```

To target a specific item, replace the For Each loop in the above code with this code:

```
For Each oUpgradeItem In oProj.ISWiUpgradeTableEntries
    If oUpgradeItem.UpgradeCode = "{12345678-1234-1234-1234-123456789012}" Then
        oUpgradeItem.Language = "1033"
    End If
Next
```

Applies To

- ISWiProject

Automation Objects for Advanced UI and Suite/Advanced UI Projects



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Each InstallShield automation object represents an installation element such as a feature, a component, or a file. This section describes each of the InstallShield automation objects.

The highest-level object in the automation interface is the ISWiProject object. The first thing you must do to access the interface is create this object. Then, you can use its objects, methods, and properties to modify your project.

ISWiProject Object (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiProject contains the interface for opening, closing, and saving an Advanced UI or Suite/Advanced UI project (.issuite) file. When InstallShield is installed on your system, you can instantiate a copy of ISWiProject with the following VB and VBScript syntax:

```
Set m_ISWiProject = CreateObject("ISWiAutoSuiteAutomation Interface Version.ISWiProject")
```

To open an installation project, call the OpenProject method, as follows:

```
' Build path to the .ism file
strFile = "C:\InstallShield 2016 Projects\Test.issuite"
m_ISWiProject.OpenProject strFile
```



Important *• If you are using the automation interface on a 64-bit machine, you may need to load the automation interface through a 32-bit executable file. Otherwise, you may encounter an error when you instantiate a copy of ISWiProject, or any other object. For more information, see Using the Automation Interface on a 64-Bit System.*

Members

The following members of the ISWIProject object are available in the automation interface. Note that some methods, properties, and collections are available only for a particular project type.

Table 12-1 • ISWiProject Object Members

Name	Type	Description
ActiveLanguage	Read-Write String Property	Gets or sets the default language of the project. Specify a single language as a language identifier; for example: `project.ActiveLanguage = "1033";`
AddExitCondition	Method	Adds an exit error message that you want the Advanced UI or Suite/Advanced UI installation to display under various conditions before ending the installation.
AddLanguage	Method	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Adds a language to the project.
AddPathVariable	Method	Adds a path variable to the project.
AddProperty	Method	Adds an Advanced UI or Suite/Advanced UI property with a specified name to the project
AddSetupFile	Method	Adds a support file with the specified name to the project.
AddSuiteAction	Method	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Adds an action to the project.
AddSuiteFeature	Method	Adds a feature to the project or as a subfeature of an existing feature.
AddSuitePackage	Method	Adds a package to the project along with the specified target file and options for adding additional files.
AddSuiteRelease	Method	Adds a release to the project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Type	Description
AddSuiteTransaction	Method	Adds a Windows Installer transaction to the project. Once your project contains a transaction, you can add .msi and .msp packages to the transaction.
ArpComments	Read-Write String Property	Gets or sets comments about your product. This information is displayed for your product's entry in Programs and Features.
ArpDisableChange	Read-Write Boolean Property	Gets or sets the value of the Disable Change Button setting in the General Information view. This setting is where you specify whether you want to disable the Change button for your product's entry in Programs and Features. The Change button enables end users to change installation options after the product has been installed. End users can remove or add features as needed.
ArpDisableRegistration	Read-Write Boolean Property	Inhibits or allows registration of the product in Programs and Features. If registration is disabled, the other Arp properties are ignored. Available values are: • **False**—Your Advanced UI or Suite/Advanced UI product's entry is displayed on the target system in Programs and Features. This is the default value. • **True**—Your Advanced UI or Suite/Advanced UI product's entry is not displayed on the target system in Programs and Features. The end user cannot use Programs and Features to remove the product, perform maintenance, or view support information. If you select this option, the other Arp properties are irrelevant.
ArpDisableRemove	Read-Write Boolean Property	Gets or sets the value of the Disable Remove Button setting in the General Information view. This setting is where you specify whether you want to disable the Remove button for your product's entry in Programs and Features. The Remove button enables end users to remove the product by clicking one button, which runs your uninstaller with a reduced user interface. If the end user clicks the Remove button to remove your product, actions in the User Interface sequence of the project are executed.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Type	Description
ArpHelpContact	Read-Write String Property	Gets or sets the name of the person or department that end users should contact for technical support. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Programs and Features.
ArpHelpTelPhone	Read-Write String Property	Gets or sets the technical support phone number for your product. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Programs and Features.
ArpHelpUrl	Read-Write String Property	Gets or sets the URL that you would like end users to visit for technical support information for your product. This URL is displayed for your product's entry in Programs and Features.
ArpIcon	Read-Write String Property	Gets or sets the path on your development system to the .ico, .exe or .dll file that contains the icon resource that you want to be used for your product's entry in Programs and Features. If you want to use the icon in a file that is present on target systems (for example, in a file that is installed by one of the packages in the Advanced UI or Suite/Advanced UI installation), you can use a directory property as part of the path for the icon file. If you leave this property empty, InstallShield uses the icon that is specified in the Setup.exe Icon File setting on the Setup.exe tab in the Releases view.
ArpIconIndex	Read-Write Integer Property	If the icon file that you specify contains more than one icon resource, you can use the ArpIconIndex property to specify the index. • A nonnegative integer refers to the order of the icon resources in the executable file. For example, 0 refers to the first icon in the file, 1 refers to the second icon, and 2 refers to the third icon. • Use a negative number to refer to a specific resource ID. For example, the icon index -12 refers to the icon with a resource ID of 12.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Type	Description
ArpProductUrl	Read-Write String Property	Gets or sets a general URL for your company or product—for example, http://www.mycompany.com. On some versions of Windows, the publisher name on the Support Info dialog box is a hyperlink to this URL. The Support Info dialog box is displayed when an end user clicks the support information hyperlink for your product's entry in Programs and Features.
ArpPublisher	Read-Write String Property	Gets or sets the name of the company that created the product. This information is displayed for your product's entry in Programs and Features.
ArpReadMe	Read-Write String Property	Gets or sets a valid URL for a Readme file or Customer Support page for your product. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Programs and Features.
ArpUpdateUrl	Read-Write String Property	Gets or sets a URL where end users can get information about product updates or download the latest version. On some versions of Windows, this information is displayed on the Support Info dialog box for your product's entry in Programs and Features.
CloseProject	Method	Closes an Advanced UI or Suite/Advanced UI project.
CreateProject	Method	Creates a new InstallShield project file (.issuite) for an Advanced UI or Suite/Advanced UI project.
DeleteExitCondition	Method	Removes an exit error message from the project.
DeleteLanguage	Method	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Removes a language from the project.
DeletePathVariable	Method	Removes a path variable from the project.
DeleteProperty	Method	Removes an Advanced UI or Suite/Advanced UI property from the project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Type	Description
DeleteSetupFile	Method	Removes a support file from the project.
DeleteSuiteAction	Method	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Removes an action from the project.
DeleteSuiteFeature	Method	Removes a feature from the project.
DeleteSuitePackage	Method	Removes a package or a Windows Installer transaction from the project.
DeleteSuiteRelease	Method	Removes a release from the project.
ForceUpgrade	Method	Upgrades the project from an earlier version of InstallShield to the current version.
GenerateGUID	Method	Generates a new, unique GUID that is returned as a string.
OpenProject	Method	Opens an Advanced UI or Suite/Advanced UI project so that you can modify it through the automation interface
ISWiLanguages	Collection	Contains all of the languages in the current project.
ISWiPathVariables	Collection	Contains all of the Advanced UI or Suite/Advanced UI path variables in the current project.
ISWiProperties	Collection	Contains all of the Advanced UI or Suite/Advanced UI properties in the current project.
ISWiSetupFiles	Collection	Contains all of the support files in the current project.
ISWiSuiteActions	Collection	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Contains all of the Suite/Advanced UI actions that are included in a project.

Table 12-1 • ISWiProject Object Members (cont.)

Name	Type	Description
ISWiSuiteEvent	Method	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Accesses a specific event in the project.
ISWiSuiteEvents	Collection	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Contains all of the events that are in a project.
ISWiSuiteFeatures	Collection	Contains every feature (as an ISWiSuiteFeature object) in your project. When ISWiSuiteFeatures is a member of ISWiProject, it contains all of the root-level features in the project. To retrieve a collection of subfeatures, access the Features property of the ISWiSuiteFeature object. The result is an ISWiSuiteFeatures collection that contains the immediate subfeatures of the current root-level feature. Repeat for further levels of subfeatures.
ISWiSuiteExitConditions	Collection	Contains all of the exit conditions in your project.
ISWiSuitePackages	Collection	Contains every package (as an ISWiSuitePackage object) and every Windows Installer transaction (also as an ISWiSuitePackage object) in your project.
ISWiSuiteReleases	Collection	Contains all of the releases in your Advanced UI or Suite/Advanced UI project.
ProductName	Read-Write String Property	Gets or sets the name of the product. For information on how the product name is used, see Specifying a Product Name.
ProjectType	Read-only Integer Property	Gets the project type. The valid value is: • **eptSuite (16)**—Advanced UI or Suite/Advanced UI project

Table 12-1 • ISWiProject Object Members (cont.)

Name	Type	Description
ProductVersion	Read-Write String Property	Gets or sets the version number for your product. The version must contain only numbers. It is typically in the format *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535. Note that although you can include the fourth field (*ddddd*), the installation does not use this part of the product version to distinguish between different product versions. To learn more, see Specifying the Product Version.
SaveProject	Method	Saves the Advanced UI or Suite/Advanced UI project.
SuiteGuid	Read-Write String Property	Gets or sets a GUID that uniquely identifies the Advanced UI or Suite/Advanced UI installation. To have generate a different GUID, call GenerateGUID. Since this code uniquely identifies your Advanced UI or Suite/Advanced UI installation, changing the Suite GUID after you have already distributed your release is not recommended.

AddExitCondition Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddExitCondition method adds an exit condition to the project. Using this method is similar to defining an exit condition in the General Information view in InstallShield. You can set the properties of the exit condition with the ISWiSuiteExitCondition object.

Syntax

```
AddExitCondition (Message As String)
```

Parameters

Table 12-2 • AddExitCondition Method Parameters

Parameter	Description
Message	Specify the error message that you want the Advanced UI or Suite/Advanced UI installation to display at run time if the condition that you are configuring is true. As an alternative, you can enter a string identifier whose value is the error message that you want the installation to use.

Applies To

ISWiProject

AddLanguage Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *Support for multilanguage installations is available with InstallShield Premier Edition.*

The AddLanguage method adds a language to the project. Using this method is similar to adding a language in the General Information view in InstallShield. You can set the properties of the new language with the ISWiLanguage object.

Syntax

```
AddLanguage (LangId As String)
```

Parameters

Table 12-3 • AddLanguage Method Parameters

Parameter	Description
LangId	Specify the language ID (as a string) that you want to add to the current project. For example: oProject.AddLanguage "1035"

Applies To

ISWiProject

AddPathVariable Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddPathVariable method adds a path variable to the project. Using this method is similar to adding a path variable in the Path Variables view in InstallShield. You can set the properties of the new path variable with the ISWiPathVariable object.



Important *• Predefined path variables cannot be modified or deleted. If you attempt to do so, exception error 3142 occurs. For more information, see Predefined Path Variables.*

Syntax

```
AddPathVariable (sName As String) As ISWiPathVariable
```

Parameters

Table 12-4 • AddPathVariable Method Parameters

Parameter	Description
sName	Specify the name of the path variable that you want to add.

Applies To

- ISWiProject

AddProperty Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddProperty method adds an Advanced UI or Suite/Advanced UI property with a specified name to the project.

Syntax

```
AddProperty (Name As String) As ISWiProperty
```

Parameters

Table 12-5 • AddProperty Method Parameters

Parameter	Description
Name	Specify the name of the Advanced UI or Suite/Advanced UI property that you want to add.

Example

The following Visual Basic lines demonstrate this method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("ISWiAutoSuiteAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.issuite"

Dim pProperty As ISWiProperty
Set pProperty = pProj.AddProperty ("MYPROP")
pProperty.Value = "MyPropValue"

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

AddSetupFile Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSetupFile method adds a support file with the specified name to the project. Using this method is similar to adding a path variable in the Support Files view in InstallShield. You can set the properties of the new path variable with the ISWiSetupFile object.

Syntax

```
AddSetupFile (FileName As String) As ISWiSetupFile
```

Parameters

Table 12-6 • AddSetupFile Method Parameters

Parameter	Description
FileName	Specify the fully qualified path and file name of the support file that you want to add.

Example

The following Visual Basic lines demonstrate this method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("ISWiAutoSuiteAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.issuite"
Dim pSetupFile As ISWiSetupFile

Set pSetupFile = pProj.AddSetupFile ("C:\My Files\MySupportFile.ext")

pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

AddSuiteAction Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSuiteAction method adds an action to the project.

Syntax

```
AddSuiteAction (ActionName As String, ActionType As ISWiSuiteActionType) as ISWiSuiteAction
```

Parameters

Table 12-7 • AddSuiteAction Method Parameters

Parameter	Description
ActionName	Specify the name of the action that you want to add.
ActionType	Specify the type of action. Available options are: • **esatDll (1)**—Calls a function in a DLL file. • **esatPowerShell (2)**—Runs a PowerShell script to perform system configuration tasks. • **esatExe (4)**—Launches an executable file. The Suite/Advanced UI engine waits for the action to complete before proceeding with the next part of the installation. • **esatStartExe (8)**—Launches an executable file. The Suite/Advanced UI engine runs the action simultaneously as the next part of the installation proceeds. • **esatSetProperty (16)**—Sets a Suite/Advanced UI property. • **esatIScript (32)**—Runs InstallScript code. • **esatClr (64)**—Calls a public method in a managed assembly.

Applies To

- ISWiProject

AddSuiteFeature Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSuiteFeature method creates a feature with the specified name. AddSuiteFeature is a member of ISWiProject. The new feature is added to the current project as a root-level feature, but when this method is a member of an ISWiSuiteFeature object, the new feature is added as a subfeature of the current feature.

The feature is created with the same default properties that new features have when you add them in InstallShield.

After you have added the feature, you can set its other properties and associate packages with it.

Syntax

```
AddSuiteFeature (SuiteFeatureKey As String) As ISWiSuiteFeature
```

Parameters

Table 12-8 • AddSuiteFeature Method Parameters

Parameter	Description
SuiteFeatureKey	The feature key is the same as the name of the feature that is displayed in the Features view. The name becomes the index for the new ISWiSuiteFeature object. This string must contain only alphanumeric characters, dots (.), or underscores (_), and it must begin with a letter or an underscore. FeatureKey must be unique among your project's feature names. Feature names are case-sensitive.

Example

The following Visual Basic lines demonstrate calling AddSuiteFeature to add a feature and a subfeature:

```
Dim pProject As ISWiProject

Set pProject = New ISWiProject
pProject.OpenProject "C:\MySetups\SampleApp.issuite"

Dim pFeat1, pFeat2 As ISWiSuiteFeature
Dim sFeat1Name, sFeat2Name As String

sFeat1Name = "ParentFeature"
sFeat2Name = "SubFeature"

' Add the feature ParentFeature, then SubFeature under ParentFeature
Set pFeat1 = pProject.AddSuiteFeature(sFeat1Name)
Set pFeat2 = pFeat1.AddSuiteSubFeature(sFeat2Name)

pProject.SaveProject
pProject.CloseProject
```

Applies To

- ISWiProject

AddSuitePackage Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Project-specific differences are noted where applicable.



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSuitePackage method adds a package (such as an .msi package, an InstallScript package, an .exe package, or other supported type) along with the specified target file and options for adding additional files. Using this method is similar to adding a package in the Packages view in InstallShield. You can set the properties of the new package with the ISWiSuitePackage object.

Syntax

```
AddSuitePackage (PackageType As ISWiSuitePackageType, TargetFileName As String, AdditionalFiles As
    ISWiAdditionalFiles) As ISWiSuitePackage
```

Parameters

Table 12-9 • AddSuitePackage Method Parameters

Parameter	Description
PackageType	Specify the type of package that you are adding to the project. The following project types are available in Advanced UI and Suite/Advanced UI projects: • **esptMsi (1)**—Windows Installer package (.msi) • **esptMsp (2)**—Windows Installer patch (.msp) • **esptIsp (4)**—InstallScript package The following project types are available only in Suite/Advanced UI projects: • **esptTransaction (-1)**—Windows Installer transaction. This package type is available in Suite/Advanced UI projects. • **esptExe (3)**—executable package (.exe). This package type is available in Suite/Advanced UI projects. • **esptIsmMsi (5)**—Basic MSI project (.ism) • **esptIsmIsp (6)**—InstallScript project (.ism) • **esptAppx (7)**—Sideloading UWP app package (.appx) • **esptWebDeploy (8)**—Web Deploy package (.zip)
TargetFileName	Specify the file in the package (such as the .msi file) that the Advanced UI or Suite/Advanced UI installation should invoke.
AdditionalFiles	Specify the type of additional files that you want to add for the package. Available options are: • **eaftNothing (0)**—Add nothing. This package does not need any additional files. • **eaftAdjacentFiles (1)**—Add adjacent files; that is, add the files next to this package. • **eaftSubFolders (2)**—Add files in subfolders; that is, add the files below this package. • **eaftAdjacentFilesAndSubFolders (3)**—Add adjacent files and files in subfolders; that is, add the files next to and below this package.

Applies To

- ISWiProject

AddSuiteRelease Method (Advanced UI and Suite/Advanced UI)

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSuiteRelease method adds a release with the specified name to the project. Using this method is similar to adding a release in the Releases view in InstallShield. You can set the properties of the new release with the ISWiSuiteRelease object.

Syntax

```
AddSuiteRelease (ReleaseName As String) as ISWiSuiteRelease
```

Parameters

Table 12-10 • AddSuiteRelease Method Parameters

Parameter	Description
ReleaseName	Specify the name of the release that you want to add.

Applies To

- ISWiProject

AddSuiteTransaction Method (Advanced UI and Suite/Advanced UI)

Project *• This information applies to Suite/Advanced UI projects.*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSuiteTransaction method adds a Windows Installer transaction to a Suite/Advanced UI project, and it returns a package. Once your project contains a transaction, you can add .msi and .msp packages to the transaction.

Syntax

```
AddSuiteTransaction () As ISWiSuitePackage
```

Parameters

The AddSuiteTransaction method does not take any parameters.

Applies To

- ISWiProject

CloseProject Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

To close the .issuite file that you opened with OpenProject, call CloseProject.

Syntax

```
CloseProject ()
```

Return Values

CloseProject always returns 0.

Applies To

- ISWiProject

CreateProject Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The CreateProject method creates a new InstallShield project file (.issuite) for Advanced UI and Suite/Advanced UI projects.

Syntax

```
CreateProject (strISWiProjectFile As String, ByVal ProjectType As ISWiProjectType)
```

Parameters

Table 12-11 • CreateProject Method Parameters

Parameter	Description
strISWiProjectFile	Enter the fully qualified path to the .issuite file that you want to create. Include the full file name.
ProjectType	Specify the type of project that you want to create. The valid value is: • **eptSuite (16)**—Creates an Advanced UI or Suite/Advanced UI project.

Example

The following sample code creates an Advanced UI or Suite/Advanced UI project and sets its product version to 2.00.0000.

```
dim proj
set proj = CreateObject("ISWiAutoSuite22.ISWiProject")

proj.CreateProject "d:\scratch\packagetest.issuite", 16   '16: ISWiProjectType.eptSuite
proj.OpenProject "d:\scratch\packagetest.issuite", false

proj.ProductVersion = "2.00.0000"

proj.SaveProject
proj.CloseProject
```

Applies To

- ISWiProject

DeleteExitCondition Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteExitCondition method removes an exit condition from the project. Using this method is similar to removing an exit condition in the General Information view in InstallShield.

Syntax

```
DeleteExitCondition (ExitCondition As ISWiSuiteExitCondition)
```

Parameters

Table 12-12 • DeleteExitCondition Method Parameters

Parameter	Description
ExitCondition	Specify the ISWiSuiteExitCondition object that you want the method to remove from your project.

Applies To

ISWiProject

DeleteLanguage Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *Support for multilanguage installations is available with InstallShield Premier Edition.*

The DeleteLanguage method removes a language from the current project. Using this method is similar to removing a language in the General Information view in InstallShield.

Syntax

```
DeleteLanguage (Language As ISWiLanguage)
```

Parameters

Table 12-13 • DeleteLanguage Method Parameters

Parameter	Description
Language	Specify the ISWiLanguage object that you want the method to remove from your project.

Applies To

- ISWiProject

DeletePathVariable Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeletePathVariable method removes a path variable with a specified name from the project.



***Important** • Predefined path variables cannot be modified or deleted. If you attempt to do so, exception error 3142 occurs. For more information, see Predefined Path Variables.*

Syntax

```
DeletePathVariable (pPathVar ISWiPathVariable) As Long
```

Parameters

Table 12-14 • AddPathVariable Method Parameters

Parameter	Description
pPathVar	Pass the ISWiPathVariable object for the path variable that you want to delete.

Applies To

- ISWiProject

DeleteProperty Method (Advanced UI and Suite/Advanced UI)



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteProperty method removes an Advanced UI or Suite/Advanced UI property with a specified name from the project.

Syntax

```
DeleteProperty(ByVal pProperty As ISWiProperty) As Long
```

Parameters

Table 12-15 • DeleteProperty Method Parameters

Parameter	Description
Property	Pass the ISWiProperty object that you want to remove.

Example

The following Visual Basic lines demonstrate this method:

```
Dim proj As ISWiAutoSuiteAutomation Interface Version.ISWiProject
Set proj = New ISWiAutoSuiteAutomation Interface Version.ISWiProject
proj.OpenProject "c:\mysetups\your project name-3.issuite"
Dim pProp As ISWiAutoSuiteAutomation Interface Version.ISWiProperty
Set pProp = proj.ISWiProperties.Item("MyProperty")
proj.DeleteProperty pProp
```

Applies To

- ISWiProject

DeleteSetupFile Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteSetupFile method removes the specified support file from the project.

Syntax

```
DeleteSetupFile (pSetupFile As ISWiSetupFile) As Long
```

Parameters

Table 12-16 • DeleteSetupFile Method Parameters

Parameter	Description
pSetupFile	Pass the ISWiSetupFile object that you want to delete.

Example

The following Visual Basic lines demonstrate this method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("ISWiAutoSuiteAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.issuite"
Dim pSetupFile As ISWiSetupFile

Set pSetupFile = pProj.ISWiSetupFiles.Item("MySupportFile.abc")
pProj.DeleteSetupFile pSetupFile

pProj.SaveProject
```

pProj.CloseProject

Applies To

- ISWiProject

DeleteSuiteAction Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteSuiteAction method removes an action from the project.

Syntax

```
DeleteAction (SuiteAction As ISWiSuiteAction)
```

Parameters

Table 12-17 • AddSuiteAction Method Parameters

Parameter	Description
SuiteAction	Specify the ISWiSuiteAction object that you want to delete.

Applies To

- ISWiProject

DeleteSuiteFeature Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteSuiteFeature method permanently removes a feature from the current project.

Syntax

```
DeleteSuiteFeature (Feature As ISWiSuiteFeature) As Long
```

Parameters

Table 12-18 • DeleteSuiteFeature Method Parameters

Parameter	Description
Feature	Specify the ISWiSuiteFeature object that you want to remove.

Applies To

- ISWiProject

DeleteSuitePackage Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteSuitePackage method removes a package or a Windows Installer transaction from the current project.

Syntax

```
DeleteSuitePackage (Package As ISWiSuitePackage) As Long
```

Parameters

Table 12-19 • DeleteSuitePackage Method Parameters

Parameter	Description
Package	Specify the ISWiSuitePackage object that you want to remove.

Applies To

- ISWiProject

DeleteSuiteRelease Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteSuiteRelease method removes a release from the Advanced UI or Suite/Advanced UI project.

Syntax

```
DeleteSuiteRelease (SuiteRelease As ISWiSuiteRelease)
```

Parameters

Table 12-20 • DeleteSuiteRelease Method Parameters

Parameter	Description
SuiteRelease	Pass the ISWiSuiteRelease object that you want to delete.

Applies To

- ISWiProject

ForceUpgrade Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ForceUpgrade method upgrades the project from an earlier version of InstallShield to the current version.

Syntax

```
ForceUpgrade (ISWiProjectFile As String)
```

Parameters

Table 12-21 • ForceUpgrade Method Parameters

Parameter	Description
ISWiProjectFile	Enter the fully qualified path to the Advanced UI or Suite/Advanced UI project file (.issuite) that you want to upgrade. Include the full file name.

Applies To

- ISWiProject

GenerateGUID Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*

- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The GenerateGUID method generates a new, unique GUID that is returned as a string.

Syntax

```
GenerateGUID ()
```

Applies To

- ISWiProject

ISWiSuiteEvent Method (Advanced UI and Suite/Advanced UI)



***Project** • This information applies to Suite/Advanced UI projects.*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteEvent accesses a specific event in the project.

Syntax

```
ISWiSuiteEvent (EventType as ISWiSuiteEventType) as ISWiSuiteEvent
```

Parameters

Table 12-22 • ISWiSuiteEvent Method Parameters

Parameter	Description
Type	Specify the type of event. Available options are: • **esetOnBegin (0)**—The OnBegin event is the first event in a Suite/Advanced UI installation. Actions that are scheduled for this event are executed during all modes (install, maintenance, repair, etc.) unless the conditional statement that is defined for the action evaluates as false. • **esetOnResuming (1)**—The OnResuming event launches its actions when a Suite/Advanced UI installation is resuming after the target machine has been restarted. • **esetOnStaging (2)**—The OnStaging event launches its actions immediately before the Suite/Advanced UI installation is staged; that is, before the Suite/Advanced UI installation downloads its package (if applicable), extracts its packages from the **Setup.exe** file (if applicable), and copies its packages to a directory that the end user specifies on the BrowseStageFolder wizard page or by setting the **ISRootStagePath** property through the command line. • **esetOnStaged (3)**—The OnStaged event launches its actions after the Suite/Advanced UI installation has downloaded its packages (if applicable), extracted them from the **Setup.exe** file, and copied them to the staging directory. • **esetOnPackagesConfiguring (4)**—The OnPackagesConfiguring event launches its actions immediately before the packages in the Suite/Advanced UI installation are being installed, modified, repaired, or removed. • **esetOnPackagesConfigured (5)**—The OnPackagesConfigured event launches its actions when the packages in the Suite/Advanced UI installation have been installed, modified, repaired, or removed.
Type (cont.)	• **esetOnRebooting (6)**—The OnRebooting event launches its actions immediately before the Suite/Advanced UI installation triggers a restart of the machine. • **esetOnEnd (7)**—The OnEnd event launches its actions immediately before the end of a successful Suite/Advanced UI installation. If the end user cancels the Suite/Advanced UI installation or the installation encounters an error, the OnEnd event does not run. • **esetOnPackageConfiguring (8)**—This event launches the actions that the Suite/Advanced UI installation should launch before installing, removing, repairing, or modifying a package. • **esetOnPackageConfigured (9)**—This event launches the actions that the Suite/Advanced UI installation should launch after installing, removing, repairing, or modifying a package. For more information, see Types of Events in a Suite/Advanced UI Installation.

Applies To

- ISWiProject

OpenProject Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

To open your Advanced UI or Suite/Advanced UI project so that you can modify it through the automation interface, call OpenProject with the fully qualified path to your .issuite file. This method is equivalent to loading a project in InstallShield.

Note • *The OpenProject method fails if the project is open in InstallShield.*

Syntax

```
OpenProject (strISWiProjectFile As String, Optional ByVal bReadOnly As Boolean) As Long
```

Parameters

Table 12-23 • OpenProject Method Parameters

Parameter	Description
strISWiProjectFile	Enter the fully qualified path to the project file (.issuite).
bReadOnly	Determines whether the project is opened with write privileges. True opens the project as a read-only file. The default value for this optional parameter is False.

Return Values and Errors

Table 12-24 • OpenProject Method Return Values and Errors

Return Value	Description
0	The project was successfully opened.
1	The project was opened, but it is locked by some other process.

Table 12-24 • OpenProject Method Return Values and Errors (cont.)

Return Value	Description
2	Attempted to open project for write access, but project opened in read-only mode.
Error Code 1100	Unable to open file: %1. Ensure that the file is a valid InstallShield project.

Example

The following Visual Basic lines demonstrate this method:

```
Dim pProj As ISWiProject
Set pProj = CreateObject("ISWiAutoSuiteAutomation Interface Version.ISWiProject")
pProj.OpenProject "C:\MySetups\Project1.issuite"
' Modify project here.
pProj.SaveProject
pProj.CloseProject
```

Applies To

- ISWiProject

SaveProject Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

To save any changes that you have made to your Advanced UI or Suite/Advanced UI project file through the automation interface, call the SaveProject method.

Syntax

```
SaveProject ()
```

Return Values and Errors

Table 12-25 • SaveProject Method Values

Return Value	Description
ERROR_SUCCESS (0)	Project is saved.
Error Code 1104	Unable to save file: %1. The project is open in read-only mode or is being used by another process.

Applies To

- ISWiProject

ISWiLanguage Object (Advanced UI and Suite/Advanced UI)

Edition • *Support for multilanguage installations is available with InstallShield Premier Edition.*

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiLanguage object represents a language whose string entries are included in the project; it provides access to the translated strings.

To include or exclude languages, use the IsIncluded property on each relevant ISWiLanguage object.

Members

Table 12-26 • ISWiLanguage Object Members

Name	Type	Description
AddStringEntry	Method	Adds a new string entry to a language.
DeleteStringEntry	Method	Removes a string entry from the project.
Id	Read-Only String Property	Gets the identifier of the language.
IsIncluded	Read-Write Boolean Property	Includes or excludes a language, or checks its current status.
Name	Read-Only String Property	Gets the name of the current language.

Applies To

- ISWiProject

AddStringEntry Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddStringEntry method adds a new string entry to a language.

Syntax

```
AddStringEntry (StringId) As ISWiStringEntry
```

Parameters

Table 12-27 • AddStringEntry Method Parameters

Parameter	Description
StringId	Specify the identifier for the string entry.

Applies To

- ISWiLanguage

DeleteStringEntry Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteStringEntry method removes a string entry from the project.

Syntax

```
DeleteStringEntry (StringEntry As ISWiStringEntry)
```

Parameters

Table 12-28 • DeleteStringEntry Method Parameters

Parameter	Description
StringEntry	Specify the ISWiStringEntry object that you want to remove from the project.

Applies To

- ISWiLanguage

ISWiPathVariable Object (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*

- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiPathVariable object represents a path variable that is included in the project. Path variables help make projects' build paths more portable across machines.

To add or remove path variables, call ISWiProject.AddPathVariable or ISWiProjectDeletePathVariable.



***Important** • Predefined path variables cannot be modified or deleted. If you attempt to do so, exception error 3142 occurs. For more information, see Predefined Path Variables.*

Members

Table 12-29 • ISWiPathVariable Object Members

Name	Type	Description
Name	Read-Write String Property	Gets or sets the name of the path variable.
PathVarType	Read-Write Integer Property	This enumerated integer property gets or sets the type of the path variable. Available values are: • **epvtPreDefined (1)**—Predefined path variables are variables that are set to certain standard Windows directories. These variables cannot be renamed or modified, but they can be used in your installation project to point to predefined directories. • **epvtCustom (2)**—Custom path variables are sometimes referred to as *standard path variables* or *user-defined variables*. These variable types are defined through InstallShield. These variables do not rely on any outside sources, such as the registry or system paths. • **epvtEnvironment (4)**—Environment path variables are based on the values of your system's environment variables. You can set an environment path variable in your project to an existing environment variable on your machine. • **epvtRegistry (8)**—Registry path variables enable you to define your own variables based on the default value of a specified registry key. The registry key must already be present when you set a path variable to the value of the key.
TestValue	Read-Write String Property	Gets or sets the test value of the path variable.

Table 12-29 • ISWiPathVariable Object Members (cont.)

Name	Type	Description
Value	Read-Write String Property	Gets or sets the value of the path variable. **For Standard Path Variables** For standard variables, enter the directory to which you would like your variable to point.  ***Note*** *• You can also refer to other path variables in the defined value by enclosing the referenced path variable name in angled brackets. For example, if you have a path variable called* `MyRoot` *with a value of* **C:***, you can refer to it in a path variable definition for another variable, such as* `Games`*. The actual path for the* `Games` *variable might be* **C:\Programs\GameFiles,** *but you can define* `Games` *as* **<MyRoot>\Programs\GameFiles***. However, if you attempt to self-reference a path variable, the literal string is used instead. For example, defining* `Games` *as* **<MyRoot>\Programs\<Games>** *actually results in* `Games` *defined as* **C:\Programs\<Games>**.

Value (cont.)		**For Registry Path Variables**
		Enter the complete registry key, with the final "subkey" being the name of the value whose data contains the folder. For example, define *`MyRegVar`* as follows:
		`HKEY_LOCAL_MACHINE\SOFTWARE\TestKey\TestValue`
		Assume that TestKey has the following subkey and values:
		`[HKEY_LOCAL_MACHINE\SOFTWARE\TestKey]`
		`@="C:\\MyPath1"`
		`"TestValue"="C:\\MyPath2"`
		`[HKEY_LOCAL_MACHINE\Software\TestKey\TestValue]`
		`@="C:\\MyPath3"`
		Even though HKEY_LOCAL_MACHINE\SOFTWARE\TestKey has a subkey called TestValue, *`MyRegVar`* points to the value TestValue, and the current value will be **C:\MyPath2**. (Note, however, that if a value named TestValue does not exist, InstallShield reads the default value (**C:\MyPath3**) of the subkey HKEY_LOCAL_MACHINE\SOFTWARE\TestKey\TestValue.)
		For Environment Path Variables
		For environment path variables, enter the name of the variable as it appears on the Environment dialog box.

Applies To

- ISWiProject

ISWiProperty Object (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiProperty object represents an Advanced UI or Suite/Advanced UI property. You can retrieve a property by specifying an item in the ISWiProperties collection.

Members

Table 12-30 • ISWiProperty Object Members

Name	Type	Description
Comments	Read-Write String Property	Gets or sets an internal note about the string entry. The comments are not displayed at run time.
FormatPropertyValue	Read-Write Boolean Property	Gets or sets the check box in the Formatted column in the Property Manager view. The check box that lets you indicate whether the value is a formatted expression that contains property names, environment variable references, or other special strings that you want to be resolved at run time. The run-time resolution occurs early at run time, before the installation executes the actions in the OnBegin event. To leave the property value as is at run time, set this property equal to False. Square brackets and other characters have no special meaning. To replace the formatted expressions at run time, set this property equal to True. To learn about the syntax that is available for formatted expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
Name	Read-Write String Property	Gets or sets the name of the Advanced UI or Suite/Advanced UI property.
Value	Read-Write String Property	Gets or sets the value of the Advanced UI or Suite/Advanced UI property.

Applies To

- ISWiProject

ISWiSetupFile Object (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSetupFile object represents a support file in your project.

To add or remove support files, call ISWiProject.AddSetupFile or ISWiProject.DeleteSetupFile.

Members

Table 12-31 • ISWiSetupFile Object Members

Name	Type	Description
FileName	Read-only String Property	Gets the name of the support file (without the path).
Language	Read-Write String Property	Gets the string that identifies the language identifier for the support file; for example: `m_pFile.Language = "1033"`
Path	Read-only String Property	Gets the fully qualified path (including the file name) of the support file's source location. Instead of hard-coding a path, you can use a path variable that is defined in the Path Variables view or part of the ISWiPathVariables collection. At build time, InstallShield replaces the path variable with the appropriate value.
Target	Read-Write String Property	Gets or sets the support file's target directory within **SETUPSUPPORTDIR**. The following line specifies the support file to be stored in the **MyFolder1\MyFolder2** folder under **SETUPSUPPORTDIR**: m_pFile.Target = "MyFolder1\MyFolder2"

Applies To

- ISWiProject

ISWiSuiteAction Object (Advanced UI and Suite/Advanced UI)



***Project** • This information applies to Suite/Advanced UI projects.*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteAction object represents an action in the Suite/Advanced UI project.

To add or remove actions, call ISWiProject.AddSuiteAction or ISWiProject.DeleteAction. Schedule actions by calling ISWiSuiteEvent.AddSuiteAction Ref or ISWiSuiteEvent.DeleteSuiteActionRef.

Note that some ISWiSuiteAction properties are applicable to only certain action types.

Members

Table 12-32 • ISWiSuiteAction Object Members

Name	Type	Description
AbortCode	Read-Write String Property	***Note*** • *This property applies to .exe actions.* Gets or sets a comma-delimited list of the codes that the action could return when the installation should be aborted. Note that if the action cannot be launched or loaded, the installation is aborted. This property is available if the value of the action's WaitForExit property is True.
ActionName	Read-Only String Property	Gets or sets the name of the action.
Arguments	Read-Write String Property	***Note*** • *This property applies to .exe actions.* If appropriate, gets or sets the command-line parameters that should be used with the action. Separate multiple parameters with spaces.
CancelCode	Read-Write String Property	***Note*** • *This property applies to .exe actions.* Gets or sets a comma-delimited list of the codes that the action could return when the installation should be cancelled. This property is available if the value of the action's WaitForExit property is True.
Class	Read-Write String Property	***Note*** • *This property applies to managed-code actions.* Gets or sets the name of the class for the managed method that you want the selected action to call. To learn more, see Working with a Managed-Code Action in a Suite/Advanced UI Installation.

Table 12-32 • ISWiSuiteAction Object Members (cont.)

Name	Type	Description
DefaultResponse	Read-Write Integer Property	 ***Note*** • *This property applies to .exe actions.* Gets or sets a string that describes how you want the installation to respond if the code that the action returns is not specified for the IgnoreCode, AbortCode, RebootCode, or CancelCode properties. Available values are: • **esadcIgnore (0)**—The installation proceeds with the next part of the installation. This is the default option. • **esadcAbort (1)**—The installation is aborted. • **esadcReboot (2)**—The target system is restarted. • **esadcCancel (3)**—The installation is cancelled. This property is available if the value of the action's WaitForExit property is True.
FormatPropertyValue	Read-Write Boolean Property	 ***Note*** • *This property applies to actions that set a property.* Gets or sets the value of the Format Property Value setting in the Events view. This setting lets you indicate whether the value that you enter in the Property Value setting (or through the PropValue property in the automation interface) is a formatted expression that contains property names, environment variable references, or other special strings that you want to be resolved at run time. Available options are: • **True**—Replace the formatted expressions at run time. • **False**—Leave the property value as is at run time. The value that you are entering does not include formatted expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/ Advanced UI Installations Resolve at Run Time.
FunctionName	Read-Write String Property	 ***Note*** • *This property applies to DLL and InstallScript actions.* For a DLL action: Gets or sets the name of the function in the DLL that you would like to call. For an InstallScript action: Gets or sets the InstallScript function that you want to call.

Table 12-32 • ISWiSuiteAction Object Members (cont.)

Name	Type	Description
IgnoreCode	Read-Write String Property	***Note*** • *This property applies to .exe actions.* Gets or sets a comma-delimited list of the codes that the action could return when the installation should proceed with the next part of the installation. This property is available if the value of the action's WaitForExit property is True.
Method	Read-Write String Property	***Note*** • *This property applies to managed-code actions.* Gets or sets the public method that you want the selected action to call. To learn more, see Working with a Managed-Code Action in a Suite/Advanced UI Installation.
PropName	Read-Write String Property	***Note*** • *This property applies to actions that set a property.* Gets or sets the property name.
PropValue	Read-Write String Property	***Note*** • *This property applies to actions that set a property.* Gets or sets the value of the property. If you select True for the FormatPropertyValue property, the value that you enter can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
RebootCode	Read-Write String Property	***Note*** • *This property applies to .exe actions.* Gets or sets a comma-delimited list of the codes that the action could return when the target system should be restarted. This property is available if the value of the action's WaitForExit property is True.

Table 12-32 • ISWiSuiteAction Object Members (cont.)

Name	Type	Description
RequireAdminPrivileges	Read-Only Boolean Property	Gets or sets the value of the Requires Administrative Privileges setting in the Events view. This setting indicates whether this action requires administrative privileges on systems that have Windows Vista or later or Windows Server 2008 or later. To learn more, see Minimizing the Number of User Account Control Prompts During Advanced UI and Suite/Advanced UI Installations.
SourcePath	Read-Write String Property	 ***Note*** • *This property applies to PowerShell actions.* Gets or sets the source path to the PowerShell script (.ps1) that you want to use for your action. If the file is not present on target systems at run time, and if none of the packages in your Suite/Advanced UI installation are going to install it, you can reference a file that is included in your project as a support file. At run time, the file is available on the target system only during your product's installation process. Instead of hard-coding a location, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.
TargetFile	Read-Write String Property	 ***Note*** • *This property applies to .exe, DLL, and managed-code actions.* Gets or sets the run-time path to the file that you want to use for your action. If the file is not present on target systems at run time, and if none of the packages in your Suite/Advanced UI installation are going to install it, you can reference a file that is included in your project as a support file. At run time, the file is available on the target system only during your product's installation process.

Table 12-32 • ISWiSuiteAction Object Members (cont.)

Name	Type	Description
Type	Read-Only Integer Property	Gets the type of action. Available value are: • **esatDll (1)**—Calls a function in a DLL file. • **esatPowerShell (2)**—Runs a PowerShell script to perform system configuration tasks. • **esatExe (4)**—Launches an executable file. • **esatSetProperty (16)**—Sets a Suite/Advanced UI property. • **esatIScript (32)**—Runs InstallScript code. • **esatClr (64)**—Calls a public method in a managed assembly.
Verb	Read-Write String Property	 ***Note*** • *This property applies to .exe actions.* Gets or sets the verb that should be used with the selected action's file. If you leave this property blank, the verb that is the default one on the target system for this file is used. If the default verb is not available, the *open* verb is used. If neither verb is available, the first verb that is listed in the registry is used. The use of the runas verb is not recommended. If the action requires administrative privileges, select True for the action's RequireAdminPrivileges property. For more information, see Minimizing the Number of User Account Control Prompts During Advanced UI and Suite/Advanced UI Installations.
WaitForExit	Read-Write Boolean Property	 ***Note*** • *This property applies to .exe actions.* Gets or sets the value of the Wait for Exit setting in the Events view. This setting is where you indicate whether you want the installation to wait for the action to complete before proceeding with the next part of the installation. Available options are: • **True**—The installation waits for the action to complete before resuming. This is the default option. • **False**—The installation runs the action simultaneously as the next part of the installation proceeds. If you select True for this setting, you can use the IgnoreCode, AbortCode, RebootCode, and CancelCode properties to specify the various codes that the action may return.

Table 12-32 • ISWiSuiteAction Object Members (cont.)

Name	Type	Description
Window	Read-Write Integer Property	***Note*** • *This property applies to .exe actions.* Gets or sets the initial window state of the .exe action when it is launched. Available options are: • **esawShow (0)**—Display the window in its current size and position. • **esawHide (1)**—Hide the window. • **esawMin (2)**—Display the window as minimized. • **esawMax (3)**—Display the window as maximized. • **esawNormal (4)**—Display the window in its original size and position.

Applies To

- ISWiProject

ISWiSuiteActionRef Object (Advanced UI and Suite/Advanced UI)

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteActionRef object represents a reference to the scheduling of an action during a Suite/Advanced UI event. You can schedule an action multiple times—even in a single event.

To schedule actions, call ISWiSuiteEvent.AddSuiteActionRef. To remove the references to the scheduling of actions, call ISWiSuiteEvent.DeleteSuiteActionRef.

Members

Table 12-33 • ISWiSuiteActionRef Object Members

Name	Type	Description
DisplayText	Read-Write String Property	 ***Note*** • *This property applies to .exe, DLL, PowerShell, InstallScript, and managed-code actions.* Gets or sets the text that describes the selected action. For example, if the action is running a script that configures the target system, you could enter the following string: `Configuring the system` If the installation launches the action when the InstallationProgress wizard page is displayed, the text that you enter is displayed on the wizard page.
MoveTo	Method	Reschedules a Suite/Advanced UI action to occur directly after the specified other action.
Name	Read-only String Property	Gets the name of the action that is scheduled.

Applies To

- ISWiProject

MoveTo Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The MoveTo method reschedules a Suite/Advanced UI action to occur directly after the specified other action.

Syntax

```
MoveTo (PrecedingActionRef As ISWiSuiteActionRef)
```

Parameters

Table 12-34 • MoveTo Method Parameters

Parameter	Description
PrecedingActionRef	Specifies the action that precedes the action that you want to reschedule. The default value of Nothing indicates the top position.

Applies To

- ISWiSuiteActionRef

ISWiSuiteCondition Object (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteCondition object represents the following types of conditions:

- Exit condition (used for the Condition property of the ISWiSuiteExitCondition object)
- Feature condition (used for the Condition property of the ISWiSuiteFeature object)
- Detection condition (used for the DetectionCondition property of the ISWiSuitePackage object)
- Eligibility condition (used for the EligibleCondition property of the ISWiSuitePackage object)

The ISWiSuiteCondition object can also exist in the Conditions collection of an All, Any, or None condition group.

To add or remove conditions from a condition group, call AddCondition, AddGroup, AddExtensionCondition, or DeleteCondition.

Members

Table 12-35 • ISWiSuiteCondition Object Members

Name	Type	Description
AddCondition	Method	Adds a condition to a condition group.
AddExtensionCondition	Method	Adds an extension condition to a condition group.
AddGroup	Method	Adds an All, Any, or None operator to a setting for a condition group.
AddParameter	Method	Adds a parameter for an extension condition.
ChangeGroupType	Method	Changes the All, Any, or None operator in a condition group.
DeleteCondition	Method	Deletes a condition object.
DeleteParameter	Method	Deletes a parameter for an extension condition.
ISWiSuiteConditionParameters	Collection	Contains all of the settings for a conditional statement.
ISWiSuiteConditions	Collection	Contains all of the conditional statements under a condition group.
Move	Method	Moves a condition or a condition group within a condition tree.
Name	String Property: Read-Write for Extension Conditions; Read-only for all other conditions	For extension conditions: Gets or sets the part of the name of the DLL entry points. For other condition types: Gets the name of the condition.
Resource	Read-Write String Property	Gets or sets the source of the condition. For extension conditions, the value of this property is set to the name of the DLL file. For the other types of built-in conditions, the value of this property is read-only.

Table 12-35 • ISWiSuiteCondition Object Members (cont.)

Name	Type	Description
Type	Read-only Integer Property	Gets the type of condition, condition group, or condition check. Available options are: • **esctAll (0)**—An All condition group operates like a logical AND operation. All of the conditions that are in the All group must evaluate to true in order for the condition group to evaluate to true. • **esctAny (1)**—An Any condition group operates like a logical OR operation. If any of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to true. If none of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to false. • **esctNone (2)**—A None condition group operates like a logical NOR operation. If none of the conditions that are in the None group evaluate to true, the entire condition group evaluates to true. If any of the conditions that are in the None condition group evaluate to true, the entire condition group evaluates to false. Note that if the None condition group consists of only one conditional statement, it operates like a logical NOT operation. • **esctBuiltin (11)**—A built-in condition check has one or more subsettings. Examples of the built-in condition checks are Platform, File Exists, Eligible Package, and Feature Operation. • **esctExtension (21)**—An extension condition uses a C/C++ DLL that you created to implement your own custom condition that checks target systems for factors that the other types of condition checks do not evaluate. To identify group conditions, use the following: `pCondition.Type < 10`

Applies To

- ISWiSuiteExitCondition
- ISWiSuiteFeature
- ISWiSuitePackage

AddCondition Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*

- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddCondition method adds a condition to a condition group.

To add an extension condition to a condition group, use the AddExtensionCondition method.

Syntax

```
AddCondition (ConditionCheckName As String) As ISWiSuiteCondition
```

Parameters

Table 12-36 • AddCondition Method Parameters

Parameter	Description
ConditionCheckName	Specify the name of the type of condition check. Available options are: • **Platform**—Check target systems for one or more characteristics such as the operating system version, the architecture, and other platform details on target systems. • **FileExists**—Check target systems for the presence of a particular file. • **FileValue**—Check target systems for specific information—date, version number, or content—for a particular file. • **RegistryExists**—Check target systems for the presence of a particular registry key, and optionally a specific value name. • **RegistryValue**—Check target systems for specific information—DWORD, QWORD, string, multistring, file version, or product version—for a particular registry entry. If you do not specify a value name, the default value for the key that you specify is used for the comparison. • **PropertyValue**—Check the value of a particular built-in Advanced UI or Suite/Advanced UI property, or a property that is defined in the Property Manager view. • **MsiInstalled**—Check target systems for the presence of a product that was installed by a particular .msi package. • **MsiRelated**—Check target systems for the presence of a version of a product that you want to update, or for the presence of future versions of a product that you do not want to downgrade. In either case, the product whose presence is being checked is installed by an .msi package.

Table 12-36 • AddCondition Method Parameters (cont.)

Parameter	Description
ConditionCheckName (cont.)	• **ParcelRef**—Check whether a different package or a feature in the Advanced UI or Suite/Advanced UI installation is eligible for installation on a target system. For example, if a patch package is eligible for installation only if the base package is eligible for installation, you can create an Eligible Package condition to tie the patch's eligibility to the base package's eligibility. • **IspInstalled**—Check target systems for the presence of a product that was installed by a particular InstallScript package. • **Locale**—Match one or more locale-related settings on target systems. • **SuiteInstalled**—Check whether a particular version of an Advanced UI or Suite/Advanced UI installation is already installed on a target system. • **AppxInstalled**—Check target systems for the presence of a particular UWP app. • **ParcelAction**—Check the target state of a particular package in the Suite/Advanced UI installation. This type of condition is available only for actions that are associated with an event in the Events view or Packages view. • **FeatureAction**—Check the target state of a particular feature in the Suite/Advanced UI installation. This type of condition is available only for actions that are associated with an event in the Events view or Packages view.  ***Project*** • *The ParcelAction and FeatureAction type of conditions are available in Suite/Advanced UI projects.*

Applies To

- ISWiSuiteCondition

AddExtensionCondition Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddExtensionCondition method adds an extension condition to a condition group.

Syntax

```
AddExtensionCondition (ConditionCheckName As String, Resource as String) As ISWiSuiteCondition
```

Parameters

Table 12-37 • AddExtensionCondition Method Parameters

Parameter	Description
ConditionCheckName	Specify the name of the condition; this is part of the name of the DLL entry points.
Resource	Specify the name of the DLL file. For the other types of built-in conditions.

Applies To

- ISWiSuiteCondition

AddGroup Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddGroup method adds an All, Any, or None operator to a setting for a condition group.

Syntax

```
AddGroup (GroupType As ISWiSuiteConditionType) As ISWiSuiteCondition
```

Parameters

Table 12-38 • AddGroup Method Parameters

Parameter	Description
GroupType	Specify the operator that you are adding to a setting for a condition group. Available options are: • **esctAll (0)**—An All condition group operates like a logical AND operation. All of the conditions that are in the All group must evaluate to true in order for the condition group to evaluate to true. • **esctAny (1)**—An Any condition group operates like a logical OR operation. If any of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to true. If none of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to false. • **esctNone (2)**—A None condition group operates like a logical NOR operation. If none of the conditions that are in the None group evaluate to true, the entire condition group evaluates to true. If any of the conditions that are in the None condition group evaluate to true, the entire condition group evaluates to false. Note that if the None condition group consists of only one conditional statement, it operates like a logical NOT operation.

Applies To

- ISWiSuiteCondition

AddParameter Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddParameter method adds a parameter for an extension condition to a conditional statement.

Syntax

```
AddParameter (Name As String) As ISWiSuiteConditionParameter
```

Parameters

Table 12-39 • AddParameter Method Parameters

Parameter	Description
Name	Specify the name of the parameter that you are adding for an extension condition.

Applies To

- ISWiSuiteCondition

ChangeGroupType Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ChangeGroupType method changes the All, Any, or None operator in a condition group.

Syntax

```
ChangeGroupType (GroupType As ISWiSuiteConditionType)
```

Parameters

Table 12-40 • ChangeGroupType Method Parameters

Parameter	Description
GroupType	Specify the operator that you want to use for a condition group. Available options are: • **esctAll (0)**—An All condition group operates like a logical AND operation. All of the conditions that are in the All group must evaluate to true in order for the condition group to evaluate to true. • **esctAny (1)**—An Any condition group operates like a logical OR operation. If any of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to true. If none of the conditions that are in the Any group evaluate to true, the entire condition group evaluates to false. • **esctNone (2)**—A None condition group operates like a logical NOR operation. If none of the conditions that are in the None group evaluate to true, the entire condition group evaluates to true. If any of the conditions that are in the None condition group evaluate to true, the entire condition group evaluates to false. Note that if the None condition group consists of only one conditional statement, it operates like a logical NOT operation.

Applies To

- ISWiSuiteCondition

DeleteCondition Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteCondition method removes a condition object (a built-in condition, an extension condition, or a condition group).

Syntax

```
DeleteCondition (pCondition As ISWiSuiteCondition)
```

Parameters

Table 12-41 • DeleteCondition Method Parameters

Parameter	Description
pCondition	Specify the ISWiSuiteCondition object for the condition that you want to remove.

Applies To

- ISWiSuiteCondition

DeleteParameter Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteParameter method removes a parameter for an extension condition from a conditional statement.

Syntax

```
DeleteParameter (pParam As ISWiSuiteConditionParameter)
```

Parameters

Table 12-42 • DeleteParameter Method Parameters

Parameter	Description
pParam	Specify the ISWiSuiteConditionParameter object that you want to remove from the conditional statement for an extension condition.

Applies To

- ISWiSuiteCondition

Move Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*

- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The Move method moves a condition or a condition group within a condition tree.

Syntax

```
Move (Direction As ISWiDirection)
```

Parameters

Table 12-43 • Move Method Parameters

Parameter	Description
Direction	Specify the direction in which you want to move a condition or a condition group within a condition tree. Available options are: • **edUp (0)**—Move the item up in the condition tree. • **edDown (1)**—Move the item down in the condition tree. • **edLeft (2)**—Move the item to the left in the condition tree. • **edRight (3)**—Move the item to the right in the condition tree.

Applies To

- ISWiSuiteCondition

ISWiSuiteConditionParameter Object (Advanced UI and Suite/Advanced UI)



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteConditionParameter object represents a parameter for a condition check. It is the equivalent of the value that is configured in a condition subsetting for a conditional statement. You can retrieve a parameter value by specifying an item in the ISWiSuiteConditionParameters collection.

The ISWiSuiteConditionParameters collection contains all of the available parameters for built-in conditions. To add or remove parameters for an extension condition, call ISWiSuiteCondition.AddParameter or ISWiSuiteCondition.DeleteParameter.

Members

Table 12-44 • ISWiSuiteConditionParameter Object Members

Name	Type	Description
Content	Read-Write String Property	Gets or sets the value of the parameter.
Name	Read-Write String Property	Gets or sets the name of the parameter. The only time that it is appropriate to customize the Name property is when you are configuring an extension condition in your project.

Applies To

- ISWiSuiteCondition

ISWiSuiteDFLFilter Object (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteDFLFilter object represents a filter for a dynamic link that includes additional files that a package in an Advanced UI or Suite/Advanced UI project requires.

Members

Table 12-45 • ISWiSuiteDFLFilter Object Members

Name	Type	Description
CanMove	Method	Indicates whether a filter can move up or down in the filter criteria; the result is returned as a Boolean value.
ISWiSuiteDFLFilters	Collection	Contains all of the filters that are associated with a dynamic link for the additional files that a package requires.
MoveDown	Method	Moves a filter down below the filter that follows it.
MoveUp	Method	Moves a filter up above its preceding filter.
Pattern	Read-Write String Property	Specify the name of the dynamic folder whose files you want to include or exclude in your dynamic link, or specify the name of a dynamic file that you want to include in your dynamic link. To indicate a wild-card character, use an asterisk (*). For example, to include all .htm files, you could enter ***.htm** in an **Include File** setting. To include all files, you could enter ***.***. To refer to the root folder (the one that contains the package file) in a path, use a dot and a backslash: **.** For example, to refer to a **ReadMe.txt** file in a subfolder called **MyDirectory**, where **MyDirectory** is a subfolder of the folder that contains the package, you could use the following filter: **.\MyDirectory\ReadMe.txt** For more information, including background on how InstallShield determines which files and folders to include in and exclude from the release, see Defining Filters for a Dynamically Linked Folder in an Advanced UI or Suite/Advanced UI Project.

Applies To

- ISWiSuiteDynamicFileLink

CanMove Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The CanMove method indicates whether a filter can move up or down in the filter criteria; the result is returned as a Boolean value.

Syntax

```
CanMove (Direction As ISWiDirection)
```

Parameters

Table 12-46 • CanMove Method Parameters

Parameter	Description
Direction	Specify one of the following ISWiDirection values to check whether it is possible to move the filter in this direction within the list of filter criteria: • **edUp (0)**—Verify whether it is possible to move the filter higher in the list of filter criteria. • **edDown (1)**—Verify whether it is possible to move the filter lower in the list of filter criteria.

Applies To

- ISWiSuiteDFLFilter

MoveDown Method (Advanced UI and Suite/Advanced UI)

***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The MoveDown method moves a filter down below the filter that follows it.

Syntax

```
MoveDown ()
```

Applies To

- ISWiSuiteDFLFilter

MoveUp Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The MoveUp method moves a filter up above the filter that precedes it.

Syntax

```
MoveUp ()
```

Applies To

- ISWiSuiteDFLFilter

ISWiSuiteDynamicFileLink Object (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteDynamicFileLink object represents a dynamic link for the additional files that a package in an Advanced UI or Suite/Advanced UI project requires.

To add or remove dynamic links, call AddDynamicFileLink or DeleteDynamicFileLink.

Members

Table 12-47 • ISWiSuiteDynamicFileLink Object Members

Name	Type	Description
AddFilter	Method	Adds a filter to a dynamic link for the additional files that a package in an Advanced UI or Suite/Advanced UI project requires.
DeleteFilter	Method	Removes a filter from a dynamic link.
ISWiSuiteDFLFilters	Collection	Contains all of the filters that are associated with a dynamic link for the additional files that a package requires.
Source	Read-Write String Property	Gets or sets the path to the source folder that contains the package that you want to include in your Advanced UI or Suite/Advanced UI project. It also is the source of dynamically included files for the package. Instead of hard-coding a location, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.

Applies To

- ISWiSuiteFolder

AddFilter Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddFilter method adds a filter to a dynamic link for the additional files that a package requires.

Syntax

```
AddFilter (FilterType As ISWiSuiteDFLFilterType) As ISWiSuiteDFLFilter
```

Parameters

Table 12-48 • AddFilter Method Parameters

Parameter	Description
FilterType	Specify the type of filter that you want to add to a dynamic link. Available options are: • **esdfIncFolder (301)**—Include folders that match the specified filter. • **esdfExcFolder (302)**—Exclude folders that match the specified filter. • **esdfIncFile (303)**—Include files that match the specified filter. • **esdfExcFile (304)**—Exclude files that match the specified filter.

Applies To

- ISWiSuiteDynamicFileLink

DeleteFilter Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteFilter method removes a filter from a dynamic link for the additional files that a package requires.

Syntax

```
DeleteFilter (Filter As ISWiSuiteDFLFilter)
```

Parameters

Table 12-49 • DeleteFilter Method Parameters

Parameter	Description
Filter	Specify the ISWiSuiteDFLFilter object that you want to remove.

Applies To

- ISWiSuiteDynamicFileLink

ISWiSuiteEvent Object (Advanced UI and Suite/Advanced UI)

Project • *This information applies to Suite/Advanced UI projects.*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteEvent object represents a Suite/Advanced UI run-time event. Actions that are scheduled for a given event are executed at run time under certain conditions.

Members

Table 12-50 • ISWiSuiteEvent Object Members

Name	Type	Description
AddSuiteActionRef	Method	Schedules a Suite/Advanced UI action during a run-time event.
DeleteSuiteActionRef	Method	Removes an action from a run-time event.
ISWiSuiteActionRefs	Collection	***Project*** • *This is available in Suite/Advanced UI projects.* ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Contains all of the Suite/Advanced UI actions that are scheduled for a specific event in a package.

Applies To

- ISWiProject

AddSuiteActionRef Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSuiteActionRef method schedules a Suite/Advanced UI action during a run-time event. You can schedule an action to run multiple times during an event if necessary.

Syntax

```
AddSuiteActionRef (Action As ISWiSuiteAction)
```

Parameters

Table 12-51 • AddSuiteActionRef Method Parameters

Parameter	Description
Action	Specify the action to schedule in this event.

Applies To

- ISWiSuiteEvent

DeleteSuiteActionRef Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteSuiteActionRef method removes a Suite/Advanced UI action from a run-time event.

Syntax

```
DeleteSuiteActionRef (ActionRef As ISWiSuiteActionRef)
```

Parameters

Table 12-52 • DeleteSuiteActionRef Method Parameters

Parameter	Description
ActionRef	Specify the action as an ISWiSuiteActionRef object that you want to delete from an event.

Applies To

- ISWiSuiteEvent

ISWiSuiteExitCondition Object (Advanced UI and Suite/Advanced UI)



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteExitCondition object represents an exit condition in the Suite/Advanced UI project.

To add or remove exit conditions, call ISWiProject.AddExitCondition or ISWiProject.DeleteExitCondition.

Members

Table 12-53 • ISWiSuiteExitCondition Object Members

Name	Type	Description
Message	Read-Write String Property	Gets or sets the error message that you want the Advanced UI or Suite/Advanced UI installation to display at run time if the conditions that are defined for this message are true.
Condition	Read-only Object Property	This property accesses the ISWiSuiteCondition object. Gets the root-level condition group (Any, All, or Not) that the Advanced UI or Suite/Advanced UI installation evaluates at run time for the exit condition. Exit conditions always have a condition, so there is no need to add a condition.

Applies To

- ISWiProject

ISWiSuiteFeature Object (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteFeature object represents a root-level feature or a subfeature in an Advanced UI or Suite/Advanced UI project. You can retrieve a feature by specifying an item in the ISWiSuiteFeatures collection.

Members

Table 12-54 • ISWiSuiteFeature Object Members

Name	Type	Description
AddCondition	Method	Adds a condition to a feature without a condition.
AddSuiteSubFeature	Method	Adds a subfeature to an existing feature.
AllowSelectionChange	Read-Write Boolean Property	Gets or sets the value of the Allow UI Selection Change setting for the feature. This setting indicates whether end users should be allowed to select or deselect the feature on the InstallationFeatures wizard page.
AttachPackage	Method	Associates a package in the project with a feature.
Condition	Read-only Object Property	This property accesses the ISWiSuiteCondition object. Gets the feature's condition, if one is associated with the feature.
Cost	Read-Write Property	Gets or sets how much space (in bytes) that the feature requires on the target system.
DeleteCondition	Method	Removes a condition from a feature.
DeleteSubSuiteFeature	Method	Removes a subfeature from a feature.
Description	Read-Write String Property	Gets or sets the description for this feature.
DisplayName	Read-Write String Property	Gets or sets the name of the feature.

Table 12-54 • ISWiSuiteFeature Object Members (cont.)

Name	Type	Description
FollowParentState	Read-Write Boolean Property	This property is available for subfeatures; it is not available for root-level features. Gets or sets the value of the Follow Parent State setting for the subfeature. This setting indicates whether you want the subfeature's state to be determined by its parent feature. Available values are: • **True**—The selected subfeature will be installed whenever its parent feature is installed. • **False**—If the selected subfeature's parent feature is installed, the selected subfeature can be either installed or not installed. This is the default option. Note that if the selected subfeature's parent feature is not installed, the selected subfeature cannot be installed. This setting is especially useful if you want the subfeature to be hidden on the InstallationFeatures wizard page, but its parent feature is not hidden.
ISWiSuiteFeatures	Collection	Contains all of the subfeatures that are assigned to a feature.
ISWiSuitePackages	Collection	Contains all of the packages that are associated with a feature.
Name	Read-Write String Property	Gets or sets the name of the feature in the collection.
ReleaseFlags	Read-Write String Property	Gets or sets the release flags that are associated with this feature. Separate multiple flags with a comma. You can specify which release flags you want to use to filter the project's features in the ReleaseFlags property of ISWiRelease.
RemovePackage	Method	Removes a package from a feature.
Visible	Read-Write Boolean Property	Gets or sets the feature's Visible setting, which specifies whether the feature will be visible on the InstallationFeatures wizard page during installation.

Example

The following VBScript lines display information about a feature. Since feature can include subfeatures, the script is implemented recursively.

```
Sub WriteFeature(indent, f)
    out.WriteLine indent & f.DisplayName & " (" & f.Name & ")"
    out.WriteLine indent & "  Release Flags:" & f.ReleaseFlags
    If f.Visible Then out.WriteLine indent & "  Visible"
    If Not f.AllowSelectionChange Then out.WriteLine indent & "  Disabled"
```

```
    If f.FollowParentState Then out.WriteLine indent & "  Follow Parent"
    If Not f.Condition Is Nothing Then
        out.WriteLine indent & "  Condition:"
        WriteCondition indent & "  ", f.Condition
    End If
    out.WriteLine indent & "  Packages:"
    For Each p in f.Packages
        out.WriteLine indent & "    " & IDS(p.DisplayName)
    Next
    out.WriteLine indent & "  Child Features:"
    For Each sf in f.SubFeatures
        WriteFeature indent & "  ", (sf)
    Next
End Sub
```

Applies To

- ISWiProject
- ISWiSuiteFeature

AddCondition Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddCondition method adds a condition to a feature without a condition.

Syntax

```
AddCondition () As ISWiSuiteCondition
```

Parameters

The AddCondition method does not take any parameters.

Applies To

- ISWiSuiteFeature

AddSuiteSubFeature Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSuiteSubFeature method adds a subfeature to an existing feature in the project.

Syntax

```
AddSuiteSubFeature (InternalName As String) As ISWiSuiteFeature
```

Parameters

Table 12-55 • AddSuiteSubFeature Method Parameters

Parameter	Description
InternalName	Specify the internal name (which is displayed in the Features pane of the Feature view) of the subfeature that you want to add to the existing feature.

Applies To

- ISWiSuiteFeature

AttachPackage Method (Advanced UI and Suite/Advanced UI)



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AttachPackage method associates a package with a feature in the project.

Syntax

```
AttachPackage (Package As ISWiSuitePackage)
```

Parameters

Table 12-56 • AttachPackage Method Parameters

Parameter	Description
Package	Specify the ISWiSuitePackage object that you want to be associated with the feature.

Applies To

- ISWiSuiteFeature

DeleteCondition Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteCondition method removes a condition from a feature.

Syntax

```
DeleteCondition ()
```

Parameters

The DeleteCondition method does not take any parameters.

Applies To

- ISWiSuiteFeature

DeleteSuiteSubFeature Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteSuiteSubFeature method removes a subfeature from a feature in the project.

Syntax

```
DeleteSuiteSubFeature (SubFeature As ISWiSuiteFeature)
```

Parameters

Table 12-57 • DeleteSuiteSubFeature Method Parameters

Parameter	Description
SubFeature	Specify the ISWiSuiteFeature object that you want to remove from the feature.

Applies To

- ISWiSuiteFeature

RemovePackage Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The RemovePackage method removes a package from a feature in the project.

Syntax

```
RemovePackage (Package As ISWiSuitePackage)
```

Parameters

Table 12-58 • RemovePackage Method Parameters

Parameter	Description
Package	Specify the ISWiSuitePackage object that you want to remove from the feature.

Applies To

- ISWiSuiteFeature

ISWiSuiteFile Object (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteFile object represents a file in a package of an Advanced UI or Suite/Advanced UI project. You can change the file or its source path. The file's other properties are read-only.

Members

Table 12-59 • ISWiSuiteFile Object Members

Name	Type	Description
FullPath	Read-Only String Property	Gets the full path of the file.
LinkPath	Read-Write String Property	Gets or sets the source path to the file. Instead of hard-coding a path, you can use a path variable that is defined in the Path Variables view or part of the ISWiPathVariables collection. At build time, InstallShield replaces the path variable with the appropriate value.
Modified	Read-Only String Property	Gets the last-modified date of the file.
Name	Read-Write String Property	Gets or sets the name of the file in the package.
Size	Read-only Integer Property	Gets the size of the file.
Version	Read-Only String Property	Gets the version number of the file.

Applies To

- ISWiSuiteFolder

ISWiSuiteFolder Object (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteFolder object represents a folder of files in a package of an Advanced UI or Suite/Advanced UI project.

Members

Table 12-60 • ISWiSuiteFolder Object Members

Name	Type	Description
AddDynamicFileLink	Method	Adds a dynamic link for the additional files that a package in an Advanced UI or Suite/Advanced UI project requires.
AddFile	Method	Adds a file to a folder in the package.
AddFolder	Method	Adds a subfolder to a folder in the package.
DeleteDynamicFileLink	Method	Removes a dynamic link for the additional files that a package requires.
DeleteFile	Method	Removes a file from a folder in the package.
DeleteFolder	Method	Removes a subfolder from a folder in the package.
ISWiSuiteDynamicFileLinks	Collection	Contains every dynamic file link in a folder in a package of an Advanced UI or Suite/Advanced UI project.
ISWiSuiteFiles	Collection	Contains all of the files in a folder of a package in an Advanced UI or Suite/Advanced UI project.
ISWiSuiteFiles	Collection	Contains all of the files in the folder.
ISWiSuiteFolders	Collection	Contains all of the subfolders in the current folder.
Name	Read-Write String Property	Gets or sets the name of the folder in the package.

Applies To

- ISWiSuitePackage

AddDynamicFileLink Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddDynamicFileLink method adds a dynamic link for the additional files that a package requires.

Syntax

```
AddDynamicFileLink (SourceFolder As String)
```

Parameters

Table 12-61 • AddDynamicFileLink Method Parameters

Parameter	Description
SourceFolder	Specify the path to the source folder of dynamically included files for the package. Instead of hard-coding a location, you can use a path variable that is defined in the Path Variables view. At build time, InstallShield replaces the path variable with the appropriate value.

Applies To

- ISWiSuiteFolder

AddFile Method (Advanced UI and Suite/Advanced UI)



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddFile method adds a file to a folder in a package of an Advanced UI or Suite/Advanced UI project.

Syntax

```
AddFile (Name As String) As ISWiSuiteFile
```

Parameters

Table 12-62 • AddFile Method Parameters

Parameter	Description
Name	Specify the name of the file that you want to add to a folder in a package.

Applies To

- ISWiSuiteFolder

AddFolder Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddFolder method adds a subfolder to a folder in a package of an Advanced UI or Suite/Advanced UI project.

Syntax

```
AddFolder (Name As String) As ISWiSuiteFolder
```

Parameters

Table 12-63 • AddFile Method Parameters

Parameter	Description
Name	Specify the name of the subfolder that you want to add to a folder in a package.

Applies To

- ISWiSuiteFolder

DeleteDynamicFileLink Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteDynamicFileLink method removes a dynamic link for the additional files that a package requires.

Syntax

```
DeleteDynamicFolderLink (DynamicFileLink As ISWiSuiteDynamicFileLink)
```

Parameters

Table 12-64 • DeleteDynamicFolderLink Method Parameters

Parameter	Description
DynamicFileLink	Specify the ISWiSuiteDynamicFileLink object that you want to remove.

Applies To

- ISWiSuiteFolder

DeleteFile Method (Advanced UI and Suite/Advanced UI)



***Project** • This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteFile method removes a file from a folder in a package of an Advanced UI or Suite/Advanced UI project.

Syntax

```
DeleteFile (File as ISWiSuiteFile)
```

Parameters

Table 12-65 • DeleteFile Method Parameters

Parameter	Description
File	Specify the ISWiSuiteFile object that you want to remove.

Applies To

- ISWiSuiteFolder

DeleteFolder Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteFolder method removes a subfolder from a folder in a package of an Advanced UI or Suite/Advanced UI project.

Syntax

```
DeleteFolder (Name As ISWiSuiteFolder)
```

Parameters

Table 12-66 • DeleteFile Method Parameters

Parameter	Description
Name	Specify the ISWiSuiteFolder object that you want to remove.

Applies To

- ISWiSuiteFolder

ISWiSuiteOperation Object (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuiteOperation object represents an operation on a package.

Note that some ISWiSuiteOperation items are not applicable to all types of Advanced UI or Suite/Advanced UI package files.

Members

Table 12-67 • ISWiSuiteOperation Object Members

Name	Type	Type of Package File	Description
CmdLine	Read-Write String Property	.msi, .msp, .exe, InstallScript, Basic MSI project, InstallScript project	If applicable, gets or sets the command line that should be used to launch the target file whenever the Advanced UI or Suite/Advanced UI installation is running with a user interface. Do not include the name of the file in this property. The value of this property can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. For more information, including valid types of entries for this property, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.
Enable	Read-Write Boolean Property	Support varies	Gets or sets a value to indicate whether an operation for a package is enabled.

Table 12-67 • ISWiSuiteOperation Object Members (cont.)

Name	Type	Type of Package File	Description
ExitBehavior	Read-Write Integer Property	.exe	If a detection condition is still not met on a target system after the package has been run, either one of the following may be true: • The executable file package was not run successfully. • The detection conditions that were specified for the package were not accurate. For example, a detection condition may indicate that the executable file needs to be launched if a particular file does not exist on the target system. If the file is still missing even after the Advanced UI or Suite/Advanced UI installation has run the executable file package, it is possible that the condition was created erroneously. The ExitBehavior property indicates the behavior that should occur if one or more of the detection conditions still indicate that the package needs to be launched. Available options are: • **epseDetectPrompt (0)**—Ask whether to continue the setup. The Advanced UI or Suite/Advanced UI installation displays a message box that prompts the end user to specify whether the Advanced UI or Suite/Advanced UI installation should continue. • **epseDetectAbort (1)**—Abort the setup. The Advanced UI or Suite/Advanced UI installation ends without proceeding further. • **epseDetectIgnore (2)**—Continue the setup. The Advanced UI or Suite/Advanced UI installation essentially ignores the executable file's unmet condition and proceeds to the next package in the installation (if one is required).
ForceApplicationShutdown	Read-Write Boolean Property	.appx	If the selected sideloading app package (or any package that depends on this package) is in use at installation run time, you can require that the processes that are associated with the package are forced to shut down so that registration can proceed. Use this property to indicate whether you want to force application shutdown in that scenario.

Table 12-67 • ISWiSuiteOperation Object Members (cont.)

Name	Type	Type of Package File	Description
RebootCode	Read-Write String Property	.exe	If the selected package requires that the target system be restarted after the product is installed, this property gets or sets one or more return codes. If multiple return codes exist, each one separated by a comma. For more information on Advanced UI or Suite/Advanced UI packages that require a restart, see Restarting a Target System for an Advanced UI or Suite/Advanced UI Package.
RebootRequest	Read-Write String Property	.msi, .msp, .exe, InstallScript, Basic MSI project, InstallScript project	The installation of an Advanced UI or Suite/Advanced UI package may require that the target system be restarted, as described in Restarting a Target System for an Advanced UI or Suite/Advanced UI Package. The RebootRequest property indicates the behavior that should occur if the package requires that a target system be restarted. Available options are: • **epsrDetectExit (1)**—Allow the machine to reboot. • **epsrDetectReboot (2)**—Prompt, then exit or reboot the machine. • **epsrDetectIgnore (4)**—Ignore the reboot request. • **epsrAlwaysRebootPrompt (8)**—Always prompt, then exit or reboot the machine. • **epsrAlwaysRebootSilent (16)**—Always reboot the machine. • **epsrDetectDelay (32)**—Delay the prompt, then exit or reboot the machine.

Table 12-67 • ISWiSuiteOperation Object Members (cont.)

Name	Type	Type of Package File	Description
Silent	Read-Write String Property	.msi, .msp, .exe, InstallScript, Basic MSI project, InstallScript project	If applicable, gets or sets the command line that should be used to launch the target file whenever the Advanced UI or Suite/Advanced UI installation is running silently (that is, without a user interface). Do not include the name of the file in this property. The value of this property can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time. For more information, including valid types of entries for this property, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.
Target	Read-Write String Property	Support varies	Gets or sets the file in the package that the Advanced UI or Suite/Advanced UI installation should invoke. This property is available in the following scenarios: • When the Type property is esotInstall (1) and the type of package file is .msi, .msp, .exe, and InstallScript packages • When the Type property is esotRemove (2), esotRepair (3), or esotModify (4), and the type of package file is .exe packages

Table 12-67 • ISWiSuiteOperation Object Members (cont.)

Name	Type	Type of Package File	Description
Type	Read-only String Property	Support varies	Gets the type of operation. Available options are: • **esotInstall (1)**—The package is launched as a first-time installation. This operation is available for the following types of package files: .msi, .msp, .exe, .appx, InstallScript, Basic MSI project, and InstallScript project. • **esotRemove (2)**—The product that the package installed is removed. This operation is available for the following types of package files: .msi, .exe, .appx, InstallScript, Basic MSI project, and InstallScript project. • **esotRepair (3)**—The product that the package installed is repaired. This operation is available for the following types of package files: .msi, .exe, InstallScript, Basic MSI project, and InstallScript project. • **esotModify (4)**—The package is launched in maintenance mode to enable end users to install, remove, or modify the product. This operation is available for the following types of package files: .msi, .msp, .exe, InstallScript, Basic MSI project, and InstallScript project.

Applies To

- ISWiSuitePackage

ISWiSuitePackage Object (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSuitePackage object represents a package or a Windows Installer transaction that is included in an Advanced UI or Suite/Advanced UI project.

To add or remove packages and transactions, call ISWiProject.AddSuitePackage or ISWiProject.DeleteSuitePackage.



Important • *Note that some ISWiSuitePackage items are not applicable to all types of Advanced UI or Suite/Advanced UI package files. Use the PackageType property to determine the package type. If one of the other ISWiSuitePackage properties returns an empty value, that property is not applicable to that package type.*

Also note that the Advanced UI project type, which is available in the Professional edition of InstallShield, does not have support for most of the types of package files; the only ones that it does support are .msi, msp, and InstallScript packages.

Members

Table 12-68 • ISWiSuitePackage Object Members

Name	Type	Type of Package File	Description
AddDetectCondition	Method	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Adds a detection condition to a package in an Advanced UI or Suite/Advanced UI project.
AddEligibleCondition	Method	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Adds an eligibility condition to a package in an Advanced UI or Suite/Advanced UI project.
AddSuitePackage	Method	.msi, .msp	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Adds an .msi or .msp package to a Windows Installer transaction.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
CachePath	Read-Write String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Gets or sets where the cached package and other package files should be stored on the end user's system. You can enter a hard-coded value such as **C:\CachedFiles**, but it is recommended that you use a destination property for the path. The default value is as follows: **[LocalAppDataFolder]Downloaded Installations** The Setup.exe tab in the Releases view lets you specify whether you want to cache the packages on target systems for any packages that are run on target systems and that have a cache path defined. If you are building an uncompressed release, you may want to avoid caching packages on the target system. The value of this property can include one or more formatted expressions that contain property names, environment variable references, and other special strings; at run time, the installation expands the values of these expressions. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
CerFile	Read-Write String Property	.appx	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Gets or sets the .cer file that you are using in your sideloading package. The versions of Windows that support UWP app packages do not install a sideloading UWP app package unless they trust the source of the package. If you are using a custom certificate, such as the one that is generated in Visual Studio, the Suite/Advanced UI installation can make Windows trust the source of your package by adding the custom certificate to the certificate store of target systems.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
DeleteCondition	Method	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Removes a detection or eligibility condition from a package.
DetectCondition	Read-only Object Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	This property accesses the ISWiSuiteCondtion object. Gets or sets the conditions that the Advanced UI or Suite/Advanced UI installation should use to evaluate whether the package is already installed on target systems.
DisplayName	Read-Write String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Gets or sets the name that is displayed for the package at run time.
DynamicParcelProductConfig	Read-Write String Property	Basic MSI project	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Gets or sets the path to the product configuration that contains the release in the InstallShield project that you want to associate with the selected package.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
DynamicParcelRelease	Read-Write String Property	Basic MSI project, InstallScript project	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Gets or sets the release in the InstallShield project that you want to associate with the selected package.
DynamicParcelSource Path	Read-Write String Property	Basic MSI project, InstallScript project	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Gets or sets the source path of the Basic MSI or InstallScript project (.ism) that you are including in the Suite/Advanced UI project.
Elevate	Read-Write Boolean Property	.msi, .msp, .exe, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Gets or sets the value of the Require Elevated Privileges setting in the Packages view. This setting indicates whether the package requires elevation on Windows Vista and later and Windows Server 2008 and later systems.
EligibleCondition	Read-only Object Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	This property accesses the ISWiSuiteCondtion object. Gets or sets the conditions that the Advanced UI or Suite/Advanced UI installation should use to determine whether the target system meets the requirements that are necessary for the package to be run.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
IsPrimary	Read-Write Boolean Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	 ***Project*** • *In Suite/Advanced UI projects, this property is available for edit. In Advanced UI projects, this setting is read-only, since an Advanced UI project has support for only one primary package.* Gets or sets the Package Type setting for a package in the Packages view. This setting identifies whether the presence of this package on the target system should influence whether the Suite/Advanced UI installation runs in first-time-installation mode or maintenance mode. Available options for this property are: • **True**—The selected package is a primary package; that is, a main part of the Advanced UI or Suite/Advanced UI installation. At run time, if all of the primary packages in the installation are absent from the target system, the installation runs as a first-time installation. If any of the primary packages are present on the target system, the installation runs in maintenance mode. • **False**—The selected package is a dependency package; that is, it should not be a factor in determining which mode is used to run the Advanced UI or Suite/Advanced UI installation.
ISWiSuiteActionRefs	Collection	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Contains all of the Suite/Advanced UI actions that are scheduled for a specific package event.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
ISWiSuiteEvents	Collection	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Contains all of the events that are associated with a package in a project.
ISWiSuiteFolders	Collection	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Contains every folder in a package of your project.
ISWiSuitePackages	Collection	transaction	Contains every package (as an ISWiSuitePackage object) in a Windows Installer transaction.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
LoggingCmdLine	Read-Write String Property	.exe, InstallScript, InstallScript project	This property is available if the value of the LoggingEnable property is True. Gets or sets the value of the Logging Command Line setting in the Packages view. This setting indicates the command line that the Advanced UI or Suite/Advanced UI installation should pass to the .exe package to enable the logging. Include any appropriate supported flags. If the .exe package that you are configuring supports it, include a path that references the Advanced UI or Suite/Advanced UI property **ISLogDir**, enclosed within square brackets, in place of the path to the directory where the log file should be created. For example, to generate a log file that logs everything verbosely for an .msi package that is run by a **Setup.exe** file that InstallShield built, enter the following command line: `/v"/l*v \"[ISLogDir]FileName.log\""` The Advanced UI or Suite/Advanced UI installation replaces **[ISLogDir]** with the path to the folder that will contain the log file. The Advanced UI or Suite/Advanced UI `/log` command-line parameter lets end users specify the path to the directory for the package log files. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.
LoggingEnable	Read-Write Boolean Property	.msi, .msp, .exe, InstallScript, Basic MSI project, InstallScript project	Gets or sets the Enable Logging Support setting in the Packages view. This setting indicates whether you want the package to generate a log file if the Advanced UI or Suite/Advanced UI installation is launched from the command line with the `/log` command-line parameter. Note that if a package does not have support for logging, it is recommended that you do not enable logging support for that package. If you select Yes, configure the other logging-related properties as needed. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
LoggingMsiLogFile	Read-Write String Property	.msi, .msp, Basic MSI project	This property is available if the value of the LoggingEnable property is True. Gets or sets the value of the Logging Command Line setting in the Packages view. This setting indicates a name for the log file. Do not include a path for the file; the Advanced UI or Suite/Advanced UI `/log` command-line parameter lets end users specify the directory for the package log files. If you leave this setting blank, the name of the log file that the installation creates is *`PackageGUID`***.log**, where *`PackageGUID`* is the GUID that is assigned to the package in the Package GUID setting in the Packages view. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.
LoggingMsiLogOptions	Read-Write String Property	.msi, .msp, Basic MSI project	This property is available if the value of the LoggingEnable property is True. Gets or sets the value of the Log Options setting in the Packages view. This setting indicates the Windows Installer log `/L` flags that you want the package to use when generating the log file. For example, to generate a log file that logs everything verbosely, enter the following in this setting: *v For additional valid flags, see the /L description. If you leave this setting blank, the asterisk (*) and v flags are used to generate the log file. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
MoveDown	Method	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project, transaction	If your project has more than one package or transaction, the Packages view lists them in the same order in which the Advanced UI or Suite/Advanced UI installation launches them at run time. The MoveDown method moves a package or transaction down in the ordered list, below the item that follows it.
MoveUp	Method	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project, transaction	If your project has more than one package or transaction, the Packages view lists them in the same order in which the Advanced UI or Suite/Advanced UI installation launches them at run time. The MoveUp method moves a package or transaction up in the ordered list, above the item that precedes it.
Name	Read-Write String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Gets or sets the internal name of the package.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
Operation	Read-only Object Property	Support varies	This property accesses the ISWiSuiteOperation object. Gets the operation of the specified type. Available options are: • **esotInstall (1)**—The package is launched as a first-time installation. This operation is available for the following types of package files: .msi, .msp, .exe, .appx, InstallScript, Basic MSI project, and InstallScript project. • **esotRemove (2)**—The product that the package installed is removed. This operation is available for the following types of package files: .msi, .exe, .appx, InstallScript, Basic MSI project, and InstallScript project. • **esotRepair (3)**—The product that the package installed is repaired. This operation is available for the following types of package files: .msi, .exe, InstallScript, Basic MSI project, and InstallScript project. • **esotModify (4)**—The package is launched in maintenance mode to enable end users to install, remove, or modify the product. This operation is available for the following types of package files: .msi, .msp, .exe, InstallScript, Basic MSI project, and InstallScript project.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
PackageLocation	Read-Write String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	This property accesses the ISWiPackageLoc object. Gets or sets where the package is configured to be located. Available options are: • **eplSourceMedia (0)**—Copy from source media. That is, store the package and its files on the source media. If the Advanced UI or Suite/Advanced UI installation will be run uncompressed from fixed media—such as CD, DVD, or local network—use this option. • **eplSetupExe (1)**—Extract from **Setup.exe**. That is, compress the package and its files into Setup.exe, to be extracted at run time, if necessary. If the entire Advanced UI or Suite/Advanced UI installation must be self-contained in **Setup.exe**, use this option. Note that the Web option results in smaller installations and shorter download time; however, the Setup.exe option provides for a completely self-contained installation. • **eplWeb (2)**—Download from the Web. That is, download the package and its files (if necessary) from the URL that is specified for the package. This option is recommended if your installation will be downloaded from the Internet and you want to minimize the Advanced UI or Suite/Advanced UI package size and download time. An Advanced UI or Suite/Advanced UI package is not downloaded if the correct version is already present on the target system. Note that you can override this property for all packages when you are configuring a release.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
PackageType	Read-only String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Gets the type of package in the project: • **esptTransaction (-1)**—Windows Installer transaction • **esptMsi (1)**—Windows Installer package (.msi) • **esptMsp (2)**—Windows Installer patch (.msp) • **esptExe (3)**—executable package (.exe) • **esptIsp (4)**—InstallScript package • **esptIsmMsi (5)**—Basic MSI project (.ism) • **esptIsmIsp (6)**—InstallScript project (.ism) • **esptAppx (7)**—Sideloading app package (.appx) • **esptWebDeploy (8)**—Web Deploy package (.zip)
PackageURL	Read-Write String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Gets or sets the URL for the root folder that will contain the selected package and its folders and files. The package and any accompanying files are downloaded from this location to target systems at run time if the package needs to be launched. This property is applicable if the value of the PackageLocation property is eplWeb (2).

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
ParcelType	Read-Write String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	 ***Project*** • *In Suite/Advanced UI projects, this property is available for edit. In Advanced UI projects, this property is read-only, since an Advanced UI project has support for only one primary package.* Gets the type of package that identifies whether the presence of this package on the target system should influence whether the Suite/Advanced UI installation runs in first-time-installation mode or maintenance mode. Available options for this setting are: • **espltPrimary (0)**—The selected package is a main part of the Advanced UI or Suite/Advanced UI installation. At run time, if all of the primary packages in the installation are absent from the target system, the installation runs as a first-time installation. If any of the primary packages are present on the target system, the installation runs in maintenance mode. • **espltDependency (1)**—The selected package should not be a factor in determining which mode is used to run the Advanced UI or Suite/Advanced UI installation. For more information, see Primary Packages vs. Dependency Packages in Advanced UI and Suite/Advanced UI Projects.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
ProgressCapture	Read-Write String Property	.msi, .msp, Basic MSI project	Gets or sets the value of the Status Messages Reporting setting in the Packages view. This setting indicates what sort of status messages are available for displaying in the user interface of the Advanced UI or Suite/Advanced UI installation. Available options are: • **epsptCaptureFull (3)**—Action text. The Advanced UI or Suite/Advanced UI installation's status messages include the action text from the standard actions and custom actions in the package. • **epsptCaptureFullVerbose (4)**—Action text and action data. The Advanced UI or Suite/Advanced UI installation's status messages include the action text as well as action data from the standard actions and custom actions in the package. At run time, the Advanced UI or Suite/Advanced UI installation updates the **ISParcelStatus** property with the action text and, if applicable, the action data in the **ISParcelStatus** property, which is displayed on the InstallationProgress wizard page of the Advanced UI or Suite/Advanced UI installation by default. For example, when the InstallFiles action in an .msi package is executing, the **ISParcelStatus** property is updated with the action text "Copying new files" to inform end users about the current progress of the installation. If you choose to include action data in the status messages, the **ISParcelStatus** property may also be updated with action data such as the name, directory, and size of each file as it is being installed on the target system.
ReleaseFlags	Read-Write String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	Gets or sets the release flags that are assigned to the package. Separate multiple flags with a comma. You can use release flags to selectively include and exclude this package from different builds of your Advanced UI or Suite/Advanced UI installation.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
RootFolder	Read-only Object Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	This property accesses the ISWiSuiteFolder object. Gets the root folder that contains the package and its folders and files.
SharedParckage	Read-Write Boolean Property	.msi, .msp, .exe, .appx, InstallScript, Basic MSI project, InstallScript project	Gets or sets the value for the Shared setting in the Packages view. This setting indicates whether the package is shared with other products and should be removed only if no other products that are left behind are using this package.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
UpgradeType	Read-Write String Property	.msi, Basic MSI project	Gets or sets the value of the Minor Upgrade Handling setting in the Packages view. This setting indicates the behavior that should occur when an earlier version of the package is present on the target system. Available options are: • **epsutNone (0)**—None. The Advanced UI or Suite/Advanced UI installation launches the package without running it in minor upgrade mode. That is, the Advanced UI or Suite/Advanced UI installation does not set the **REINSTALL** or **REINSTALLMODE** properties for the package. • **epsutAuto (1)**—Automatic. The Advanced UI or Suite/Advanced UI installation launches the package in upgrade mode by setting the **REINSTALL** property to *ALL* and the **REINSTALLMODE** property to *vomus*. It does not inform the end user that the product will be upgraded. • **epsutPrompt (2)**—Ask the user. The Advanced UI or Suite/Advanced UI installation displays a secondary window that asks the end user whether they want to continue. If the end user chooses to continue, the Advanced UI or Suite/Advanced UI installation launches the package in upgrade mode. If the end user chooses to not continue, the package does not run.

Table 12-68 • ISWiSuitePackage Object Members (cont.)

Name	Type	Type of Package File	Description
WindowsFeatures	Read-Write String Property	.msi, .msp, .exe, .appx, InstallScript, Web Deploy, Basic MSI project, InstallScript project	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Gets or sets a semicolon-delimited list of one or more Windows roles and features that the selected package requires be enabled on target systems that have Windows Vista or later or Windows Server 2008 or later. The names of the roles and features should be the string that tools such as Deployment Image Servicing and Management (**DISM.exe**) and Package Manager (**Pkgmgr.exe**) use to identify the role or feature on a particular version of Windows. For a list of available Windows roles and features on each version of Windows for which you are supporting, see Microsoft TechNet. At run time, if this package is eligible to be installed and it requires one or more Windows roles and features that are disabled, the Suite/Advanced UI installation enables them before launching the package. For more information, see Enabling Windows Roles and Features During a Suite/Advanced UI Installation.

Example

The following VBScript lines demonstrate how to display information about a package in a project. Windows Installer transactions can contain multiple packages, so the script is implemented recursively.

```
Sub WritePackage(indent, p)
    If p.PackageType = -1 Then ' Transaction
        out.WriteLine indent & "Transaction:"
        For Each sp In p.ISWiSuitePackages
            WritePackage indent & "  ", sp
        Next
    Else
        out.WriteLine indent & IDS(p.DisplayName) & " (" & p.Name & ")"
        out.WriteLine indent & "Type: " & p.PackageType
        out.WriteLine indent & "Description: " & p.Description
        out.WriteLine indent & "Shared: " & p.SharedPackage
        out.WriteLine indent & "Location: " & p.PackageLocation & " " & p.PackageUrl
        out.WriteLine indent & "Release Flags: " & p.ReleaseFlags
        out.WriteLine indent & "Files:"
        WriteFolder indent & "  ", p.RootFolder
        out.WriteLine indent & "Detection Condition:"
```

```
        WriteCondition indent & "  ", p.DetectCondition
        out.WriteLine indent & "Eligibility Condition:"
        WriteCondition indent & "  ", p.EligibleCondition
    End If
End Sub
```

Applies To

- ISWiProject

AddDetectCondition Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddDetectCondition method adds a detection condition to a package in an Advanced UI or Suite/Advanced UI project.

The detection condition is a condition that the Advanced UI or Suite/Advanced UI installation should use to evaluate whether the package is already installed on target systems. If this does not change from false to true (or from true to false) at run time, the Advanced UI or Suite/Advanced UI installation assumes that the package failed to install (or remove) its payload. In addition, subsequent maintenance and remove operations (or subsequent maintenance and install operations) may not behave as expected.

Syntax

```
AddDetectCondition () As ISWiSuiteCondition
```

Parameters

The AddDetectCondition method does not take any parameters.

Applies To

- ISWiSuitePackage

AddEligibleCondition Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddEligibleConditionPackage method adds an eligibility condition to a package in an Advanced UI or Suite/Advanced UI project.

The eligibility condition is a condition that the Advanced UI or Suite/Advanced UI installation should use to determine whether the target system meets the requirements that are necessary for the package to be run. For example, if the package should be run only on 64-bit systems, you could set up an x64 platform requirement in a condition; the Advanced UI or Suite/Advanced UI installation would launch the package only on 64-bit systems. You may also want to set up an eligibility condition to prevent end users from being able to install the current package version over a future newer version.

Syntax

```
AddEligibleCondition () As ISWiSuiteCondition
```

Parameters

The AddEligibleConditionPackage method does not take any parameters.

Applies To

- ISWiSuitePackage

AddSuitePackage Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The AddSuitePackage method adds a package (an .msi package or an .msp patch) to a Windows Installer transaction in the project. It also adds the specified target file and options for adding additional files. Using this method is similar to adding a package to a transaction in the Packages view in InstallShield. You can set the properties of the new package with the ISWiSuitePackage object.

To add other types of packages to the project, see AddSuitePackage Method (Advanced UI and Suite/Advanced UI).

Syntax

```
AddSuitePackage (PackageType As ISWiSuitePackageType, TargetFileName As String, AdditionalFiles As
    ISWiAdditionalFiles) As ISWiSuitePackage
```

Parameters

Table 12-69 • AddSuitePackage Method Parameters

Parameter	Description
PackageType	Specify the type of package that you are adding to the transaction. Available options are: • **esptMsi (1)**—Windows Installer package (.msi) • **esptMsp (2)**—Windows Installer patch (.msp)
TargetFileName	Specify the file in the package (such as the .msi file) that the Advanced UI or Suite/Advanced UI installation should invoke.
AdditionalFiles	Specify the type of additional files that you want to add for the package. Available options are: • **eaftNothing (0)**—Add nothing. This package does not need any additional files. • **eaftAdjacentFiles (1)**—Add adjacent files; that is, add the files next to this package. • **eaftSubFolders (2)**—Add files in subfolders; that is, add the files below this package. • **eaftAdjacentFilesAndSubFolders (3)**—Add adjacent files and files in subfolders; that is, add the files next to and below this package.

Applies To

- ISWiSuitePackage

DeleteCondition Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The DeleteCondition method removes a detection or eligibility condition from a package in an Advanced UI or Suite/Advanced UI project.

Syntax

```
DeleteCondition (pCondition As ISWiSuiteCondition)
```

Parameters

Table 12-70 • DeleteCondition Method Parameters

Parameter	Description
pCondition	Specify the ISWiSuiteCondition object for the condition that you want to remove. The ISWiSuiteCondition object that you specify should be the condition object that is returned from the package's EligibleCondition property or its DetectionCondition property.

Applies To

- ISWiSuitePackage

MoveDown Method (Advanced UI and Suite/Advanced UI)

Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

If your project has more than one package or transaction, the Packages view lists them in the same order in which the Advanced UI or Suite/Advanced UI installation launches them at run time. The MoveDown method moves a package or transaction down in the ordered list, below the item that follows it.

Syntax

```
MoveDown ()
```

Parameters

The MoveDown method does not take any parameters.

Applies To

- ISWiSuitePackage

MoveUp Method (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

If your project has more than one package or transaction, the Packages view lists them in the same order in which the Advanced UI or Suite/Advanced UI installation launches them at run time. The MoveUp method moves a package or transaction up in the ordered list, above the item that precedes it.

Syntax

```
MoveUp ()
```

Parameters

The MoveUp method does not take any parameters.

Applies To

- ISWiSuitePackage

ISWiSuiteRelease Object (Advanced UI and Suite/Advanced UI)



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Note that some edition- and project-specific differences are noted where applicable.

The ISWiSuiteRelease object represents a release in the Releases view of InstallShield.

To add or remove releases, call ISWiProject.AddSuiteRelease or ISWiProject.DeleteSuiteRelease.

Members

Table 12-71 • ISWiSuiteRelease Object Members

Name	Type	Description
Build	Method	Build a release. Note that to get any feedback about the build process, you must subscribe to events on the release. To learn more, see Using Build Status Events.
BuildLocation	Read-Write String Property	Gets or sets the top-level directory where the release will be built.
CachePackagesLocally	Read-Write Boolean Property	Gets or sets the Cache Packages Locally setting on the Setup.exe tab in the Releases view. This setting lets you specify whether you want to cache the Advanced UI or Suite/Advanced UI packages that are run on a target system in the location that is defined for each package in the Packages view. The default value is True. If you are building an uncompressed release, you may want to specify No for the value of this setting.
CertPassword	Write-only String Property	Sets the password for the .pfx file if the file has a password. InstallShield encrypts the password and stores it in your project file (.ism). At build time, InstallShield uses the password to sign files with a .pfx file. If your certificate is protected by a password but you do not enter it in this setting, signing with a .pfx file fails.  ***Note*** • *This property is not applicable if the certificate that you are using for signing is in a certificate store. Note that if the certificate was imported into the store with password protection, you will be prompted to enter the password at build time.*
DefaultLanguage	Read-Write String Property	Gets or sets the language identifier for the default language of a given release. This property value overrides the default project language that is configured in the General Information view or the String Editor view. For more information, see Setting the Default Project Language.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
DigitalCertificateInfo	Read-Write String Property	Gets or sets the digital certificate that you want to use to sign your release. To specify a .pfx certificate on your machine, specify the path and file name. To reference a certificate store that contains the certificate, use a string in the following format: `*Store*StoreLocation:Cert Subject` **Store** refers to the name of the certificate store that contains the certificate: Available options are: • My • Root • Trust • CA **StoreLocation** refers to the location of the certificate store that contains the certificate. Available options are: • User • Machine **Cert Subject** refers to the subject of the certificate.
DigitalURL	Read-Write String Property	Gets or sets the URL that is used in your digital certificate to link to a location that you would like end users to visit to learn more about your product, organization, or company. Use a fully qualified URL—for example, **http://www.mydomain.com**.
EnableLangSelectionDialog	Read-Write Boolean Property	Gets or sets the Languages Dialog setting on the Build tab in the Releases view. This setting lets you specify whether you want your installation to display the language selection wizard page when the installation is run with a full user interface.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
LauncherPrivileges	Read-Write String Property	Gets or sets the Required Execution Level setting on the Setup.exe tab in the Releases view. This setting is where you specify the minimum level that is required by your project's **Setup.exe** file for running on Windows Vista and later platforms. The available options are: • **erelAsAdmin (2)**—**Setup.exe** requires administrative privileges to run. Administrators must authorize it; non-administrators must authenticate as an administrator. • **erelAsHighestAvail (1)**—**Setup.exe** prefers administrative privileges. Administrators must authorize it; non-administrators run it without administrative privileges. • **erelAsInvoker (0)**—**Setup.exe** does not require administrative privileges, and all users can run it without administrative privileges. **Setup.exe** does not display any UAC messages prompting for credentials or for consent. This is the default option for Advanced UI and Suite/Advanced UI projects. InstallShield embeds an application manifest in the **Setup.exe** launcher. This manifest specifies the selected execution level. Operating systems earlier than Windows Vista ignore the required execution level.
Name	Read-Write String Property	Gets or sets the internal name for the release.
PackageLocation	Read-Write Boolean Property	Gets or sets the run-time location for the packages that are configured in the Packages view of the project. Available options are: • **eplSourceMedia (0)**—Copy from source media. • **eplSetupExe (1)**—Extract from Setup.exe. • **eplWeb (2)**—Download from the Web. • **eplIndividual (3)**— Follow individual selections. To learn more, see Specifying the Run-Time Location for Advanced UI or Suite/Advanced UI Packages at the Release Level.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
Password	Read-Write String Property	Gets or sets the password for your installation. Passwords are case-sensitive. This property is used only if you the value of the PasswordProtect property is True. To learn more, see Password-Protecting an Advanced UI or Suite/ Advanced UI Installation.
PasswordProtect	Read-Write Boolean Property	Gets or sets the Password Protect Launcher setting on the Setup.exe tab in the Releases view. This setting lets you indicate whether you want to password-protect your installation. To learn more, see Password-Protecting an Advanced UI or Suite/ Advanced UI Installation.
PathVariableOverrides	Read-Write String Property	Gets or sets the value of a path variable in your project for the specified release. If this property is used to set the value of a path variable, it overrides any value that is set in the Path Variables view. Note that you can use this variable to get or override user-defined path variables, environment variables, and registry variables that are configured in the Path Variables view; however, you cannot override predefined path variables such as *<WindowsFolder>*. Following is sample Visual Basic code that overrides the value of a path variable called PathVar1: `pISWiSuiteRelease.PathVariableOverrides = "PathVar1=C:\Test1" + vbCrLf + "PathVar2=C:\test2"`
PostBuildEvent	Read-Write String Property	 ***Edition*** • *This property is available in the Premier edition of InstallShield.* Gets or sets the command that is run after InstallShield has built and signed the release. To specify more than one command, enter each command on a separate line in the string. At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one. When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands. To learn more, see Specifying Commands that Run Before, During, and After Builds.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
PreBuildEvent	Read-Write String Property	 ***Edition*** • *This property is available in the Premier edition of InstallShield.* Gets or sets the command that is run before InstallShield starts building the release. This event runs after InstallShield creates the release folder and log file, but before InstallShield starts building the release. To specify more than one command, enter each command on a separate line in the string. At build time, InstallShield runs each command in the order that they are listed. The build waits until a command finishes before proceeding to the next one. When you are entering a command, you can use any path variables and environment variables that are defined in your project, instead of using a hard-coded path. You can also use certain variables that are defined specifically for build event commands. To learn more, see Specifying Commands that Run Before, During, and After Builds.

PreProcessorDefines	Read-Write String Property	
		Project • *This information applies to Suite/Advanced UI projects.*
		Gets or sets the preprocessor variable definitions. Preprocessor variable definitions that are specified here apply only to the current release; they are not used when compiling the script for other releases. Use the following format, with no spaces before or after equals signs or commas:
		MYVARIABLE1=123,MYVARIABLE2
		Such variables can be tested in the script by #if and #ifdef statements that control the flow of the script.
		Entering the name of a preprocessor variable for this setting defines the variable. For example if you enter **MYVARIABLE** for this setting, the script commands in the following #ifdef loop are executed:
		`#ifdef MYVARIABLE`
		`  // Commands`
		`#endif`
		After you add or change a preprocessor variable definition for this setting, you must compile your installation for the addition or change to take effect.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
ProductNameOverride	Read-Write String Property	Gets or overrides the product name that was specified in the General Information view with a new value for the selected release. For information on how the product name is used, see Specifying a Product Name.
ProductVersionOverride	Read-Write String Property	Gets or overrides the product version that was specified in the General Information view with a new value for the selected release. The version must contain only numbers. The version number is typically in the format *aaa.bbb.ccccc* or *aaa.bbb.ccccc.ddddd*, where *aaa* represents the major version number, *bbb* represents the minor version number, *ccccc* represents the build number, and *ddddd* represents the revision number. The maximum value for the *aaa* and *bbb* portions is 255. The maximum value for *ccccc* and *ddddd* is 65,535. Note that although you can include the fourth field (*ddddd*), the installation does not use this part of the product version to distinguish between different product versions. To learn more, see Specifying the Product Version. The product version that you specify is displayed on the Properties dialog box for **Setup.exe**. For more information, see Customizing File Properties for the Setup Launcher.
ReleaseFlags	Read-Write String Property	Gets or sets a string that contains the release flags that you want to use to filter your features. Separate multiple flags with a comma. Release flags enable you to customize your installation by including or excluding certain items in each release. Once you have assigned release flags to features and packages in the Advanced UI or Suite/Advanced UI project, you can create a release that includes features and packages based on those flags. By default, all features and packages are included in a release. Once you specify a flag for the ReleaseFlags property (or for the Release Flags setting on the Build tab in the Releases view), only unflagged items and items that contain the specified release flag are included in your installation.  ***Note*** • *If a release does not have release flags, it will include all applicable items that have release flags. To include only unflagged items, specify a flag that does not exist. For example, you might use* ***NoFlags****. This way, only unflagged items are built into a release.*

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
SetupExeCompany	Read-Write String Property	Gets or sets the value to override the default company name for **Setup.exe**. Note that you must also select a value of True for the UseMyVersionInfo property. The company name is displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. To learn more, see Customizing File Properties for the Setup Launcher.
SetupExeCopyright	Read-Write String Property	Gets or sets a custom copyright notice for **Setup.exe**. Note that you must also select a value of True for the UseMyVersionInfo property. The copyright is displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. To learn more, see Customizing File Properties for the Setup Launcher.
SetupExeDescription	Read-Write String Property	Gets or sets a custom file description for **Setup.exe**. Note that you must also select a value of True for the UseMyVersionInfo property. The description is displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. To learn more, see Customizing File Properties for the Setup Launcher.
SetupExeVersion	Read-Write String Property	Gets or sets the value to override the default product version and file version for **Setup.exe** with your own version number. Note that you must also select a value of True for the UseMyVersionInfo property. The product version and file version are displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. Note that the file version always contains four fields. If you specify fewer than four fields for your product version, the remaining fields are filled with zeros. For example, if you specify a product version of **1.1**, the file version that is used in the version resources of **Setup.exe** is **1.1.0.0**. To learn more, see Customizing File Properties for the Setup Launcher.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
SetupFileName	Read-Write String Property	Gets or sets the file name—without the .exe file extension—that InstallShield should use for the setup launcher file that it generates at build time. If this setting is blank, InstallShield uses the default value of *Setup*, and the setup launcher file is called **Setup.exe**.
SetupIcon	Read-Write String Property	Gets or sets the release's Setup.exe Icon File setting, which specifies the fully qualified name of the file from which the executable file's icon is taken at build time. If the property's value is null, a default icon is used. By default, the icon with index 0 is used; to specify a different icon, add a comma (,) as well as the icon's index or resource ID (preceded by a minus sign) after the file name. For example, **C:\Temp\MyLibrary.dll,2** indicates the icon with an index of 2, and **C:\Temp\MyLibrary.dll,-100** indicates the icon with a resource ID of 100. Instead of hard-coding a path, you can use a path variable that is defined in the Path Variables view or part of the ISWiPathVariables collection. At build time, InstallShield replaces the path variable with the appropriate value.
SignatureDescription	Read-Write String Property	Gets or sets the signature description for the **Setup.exe** file. The description that you specify is displayed on the User Account Control (UAC) box to the right of the "Program Name:" label. The UAC dialog box opens when an end user launches the signed file and elevated privileges are required. If you leave this setting blank, InstallShield uses the name of the file without its extension as the description to the right of the "Program Name:" label on the UAC dialog box.
SignEnabled	Read-Write Boolean Property	Gets or sets the Sign Setup.exe setting on the Signing tab in the Releases view. This setting lets you indicate whether you want to sign the Advanced UI or Suite/Advanced UI installation.
SuiteGuidOverride	Read-Write String Property	Gets or overrides the GUID that uniquely identifies the Advanced UI or Suite/Advanced UI installation.
UILanguages	Read-Write String Property	Gets or sets a comma-delimited string of decimal language identifiers of the user interface languages that are includes in a release. Note that if a language is not selected in the Setup Languages setting in the General Information view of the project, it is not a valid language for the UILanguages property.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
UpdateUrl	Read-Write String Property	If you may want to make updates for your Advanced UI or Suite/Advanced UI setup launcher available for download when they are available, use this property to specify the absolute path, including the file name, for an updated Advanced UI or Suite/Advanced UI setup launcher. This support makes it possible for end users to easily obtain a newer version of your product before installing an earlier version. For more information about updates for your Advanced UI or Suite/Advanced UI installation, including requirements for the update, see Supporting Downloadable Updates for an Advanced UI or Suite/Advanced UI Installation.
UseMyVersionInfo	Read-Write Boolean Property	Gets or sets the Use Custom Version Properties setting on the Setup.exe tab in the Releases view. This setting lets you specify whether you want to override the default copyright notice and file description for **Setup.exe** with your own copyright notice and file description. If you specify a value of True, enter your own custom values for the SetupExeCopyright and SetupExeDescription properties. The copyright and description are displayed on the Properties dialog box for the setup launcher; this Properties dialog box opens when end users right-click the **Setup.exe** file and then click Properties. To learn more, see Customizing File Properties for the Setup Launcher.
UsePathVariableTestValues	Read-Write Boolean Property	If you used test values for any of your path variables, set this property to True to set those variables to their actual values at this time.
VMConfig	Read-Write String Property	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Gets or sets the name of the group of VM configuration settings that you want to use when distributing the selected release to a VM. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
VMMachinePassword	Read-Write String Property	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* If a password is specified for the VM configuration that is selected through the VMConfig property, that password is considered to be the default value for the VMMachinePassword property. To override the password for the selected release, set the VMMachinePassword property. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.
VMMachineUserName	Read-Write String Property	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* If a user name is specified for the VM configuration that is selected through the VMConfig property, that user name is considered to be the default value for the VMMachineUserName property. To override the user name for the selected release, set the VMMachineUserName property. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
VMSnapShot	Read-Write String Property	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Optionally gets or sets the name of the snapshot that you want to use for distributing the release. If a snapshot is specified for the VM configuration that is selected through the VMConfig property, the name of that snapshot is considered to be the default value for the VMSnapShot property. If no snapshot is specified for the VM configuration, InstallShield does not revert to a specific snapshot. InstallShield powers on the VM without reverting it to a specific snapshot, and copies your installation to the VM. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.
VMStageMachineCopyPath	Read-Write String Property	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* The copy path identifies the location on the VM where the release is to be distributed. The last subfolder in the path can be a path that InstallShield creates on the VM; the other folders in the path must already exist. If a destination path is specified for the VM configuration that is selected through the VMConfig property, that destination path is considered to be the default value for the VMStageMachineCopyPath property. To override the destination path for the selected release, set the VMStageMachineCopyPath property. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Table 12-71 • ISWiSuiteRelease Object Members (cont.)

Name	Type	Description
VMStagePostBuild	Read-Write Boolean Property	 ***Project*** • *This is available in Suite/Advanced UI projects.*  ***Edition*** • *The Suite/Advanced UI project type is available in the Premier edition of InstallShield.* Specify whether you want InstallShield to automatically distribute the selected release each time that it is successfully built. Distributing the release to the VM involves the following tasks: **1.** Revent the VM to the designated snapshot (if one is specified). **2.** Power on the VM. **3.** Copy the built release to the VM to make it available for testing. To learn more about VM distribution, see Distributing Releases to a Virtual Machine that InstallShield Provisions at Build Time or on Demand.

Applies To

- ISWiProject

Build Method (Advanced UI and Suite/Advanced UI)



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

To build a release in an Advanced UI or Suite/Advanced UI project, call the Build method.

Syntax

```
Build ()
```

Build Status Events

The ISWiSuiteRelease object uses the following build status events:

- StatusMessage (string Message, ref bool Cancel)
- ProgressMax (int Max, ref bool Cancel)
- ProgressIncrement (int Increment, ref bool Cancel)
- TextCancel (ref bool Cancel)
- GetDotNetLibrary (out object ???)
- WarningMessage (string Message)
- ErrorMessage (string Message)
- IsCancelledByUserAnError (ref bool IsError)
- StatusMessageEx (string Message, object Tag, ref bool Cancel)

These events occur during the build process; they provide status updates. You can set pbCancel to stop the build.

Applies To

- ISWiRelease

Automation Collections for Advanced UI and Suite/Advanced UI Projects



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

Each Advanced UI or Suite/Advanced UI automation collection is a set of objects. For example, the ISWiSuiteFeatures collection is a set of ISWiSuiteFeature objects in the current project. This section describes each of the Advanced UI or Suite/Advanced UI automation collections.

ISWiLanguages Collection



Edition *• Support for multilanguage installations is available with InstallShield Premier Edition.*



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiLanguages collection contains all of the languages in the current language.

Members

Table 12-72 • ISWiLanguages Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of string entries in the ISWiLanguages collection.
Item	Read-Only Property	Provide the index number or name of the language to retrieve the ISWiLanguage object.

Applies To

- ISWiProject

ISWiPathVariables Collection



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiPathVariables collection contains all of the Advanced UI or Suite/Advanced UI properties in the current project.

Members

Table 12-73 • ISWiPathVariables Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of path variables in the ISWiPathVariables collection.
Item	Read-Only Property	Provide the index number or name of the path variable to retrieve the ISWiPathVariable object.

Applies To

- ISWiProject

ISWiProperties Collection



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiProperties collection contains all of the Advanced UI or Suite/Advanced UI properties in the current project.

Members

Table 12-74 • ISWiProperties Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of string entries in the ISWiProperties collection.
Item	Read-Only Property	Provide the index number or name of the property to retrieve the ISWiProperty object.

Applies To

- ISWiProject

ISWiSetupFiles Collection



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The ISWiSetupFiles collection contains all of the support files in the current project.

Members

Table 12-75 • ISWiSetupFiles Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of string entries in the ISWiSetupFiles collection.
Item	Read-Only Property	Provide the index number or name of the setup file to retrieve the ISWiSetupFiles object.

Applies To

- ISWiProject

ISWiSuiteActionRefs Collection



Project *• This information applies to Suite/Advanced UI projects.*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteActionRefs is a collection that contains all of the Suite/Advanced UI actions that are scheduled for a specific event in a package.

Members

Table 12-76 • ISWiSuiteActionRefs Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteActionRefs collection.
Item	Read-Only Property	Provide the index number or name of the event to retrieve the ISWiSuiteActionRef object.

Applies To

- ISWiSuitePackage

ISWiSuiteActions Collection



Project • *This information applies to Suite/Advanced UI projects.*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteActions is a collection that contains all of the Suite/Advanced UI actions that are included in a project.

Members

Table 12-77 • ISWiSuiteActions Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteActions collection.
Item	Read-Only Property	Provide the index number or name of the action to retrieve the ISWiSuiteAction object.

Applies To

- ISWiProject

ISWiSuiteConditionParameters Collection



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteConditionParameters is a collection that contains all of the settings for a conditional statement.

Members

Table 12-78 • ISWiSuiteConditionParameters Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteConditionParameters collection.
Item	Read-Only Property	Provide the index number or name of the condition parameter to retrieve the ISWiSuiteConditionParameter object.

Applies To

- ISWiSuiteCondition

ISWiSuiteConditions Collection

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteConditions is a collection that contains all of the conditional statements under a condition group.

Members

Table 12-79 • ISWiSuiteConditions Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteConditions collection.
Item	Read-Only Property	Provide the index number of the conditional statement to retrieve the ISWiSuiteCondition object.

Applies To

- ISWiSuiteCondition

ISWiSuiteDFLFilters Collection



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteDFLFilters is a collection that contains all of the filters that are associated with a dynamic link for the additional files that a package in an Advanced UI or Suite/Advanced UI project requires.

Members

Table 12-80 • ISWiSuiteDFLFilters Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteDFLFilters collection.
Item	Read-Only Property	Provide the index number or name of the dynamic file link to retrieve the ISWiSuiteDFLFilter object.

Applies To

- ISWiSuitePackage

ISWiSuiteDynamicFileLinks Collection

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteDynamicFileLinks is a collection that contains every dynamic file link in a folder in a package of an Advanced UI or Suite/Advanced UI project.

Members

Table 12-81 • ISWiSuiteDynamicFileLinks Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteDynamicFileLinks collection.
Item	Read-Only Property	Provide the index number or name of the dynamic file link to retrieve the ISWiSuiteDynamicFileLink object.

Applies To

- ISWiSuiteFolder

ISWiSuiteEvents Collection

Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*

Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteEvents is a collection that contains the events for your project or the events that are associated with a package in your project.

Members

Table 12-82 • ISWISuiteEvents Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWISuiteEvents collection.
Item	Read-Only Property	Provide the index number of the event to retrieve the ISWISuiteEvent object.

Applies To

- ISWiProject
- ISWiSuitePackage

ISWiSuiteExitConditions Collection



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWISuiteExitConditions is a collection that contains all of the exit conditions in your project.

Members

Table 12-83 • ISWISuiteExitConditions Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWISuiteExitConditions collection.
Item	Read-Only Property	Provide the index number of the exit condition to retrieve the ISWISuiteExitCondition object.

Applies To

- ISWiProject

ISWiSuiteFeatures Collection



Project *• This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition *• The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteFeatures is a collection that contains every feature (as an ISWiSuiteFeature object) in your project.

When ISWiSuiteFeatures is a member of ISWiProject, it contains all of the root-level features in the project. To retrieve a collection of subfeatures, access the Features property of the ISWiSuiteFeature object. The result is an ISWiSuiteFeatures collection that contains the immediate subfeatures of the current root-level feature. Repeat for further levels of subfeatures.

Members

Table 12-84 • ISWiSuiteFeatures Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteFeatures collection.

Item	Read-Only Property	Provide the index number or name of the feature to retrieve the ISWiSuiteFeature object. For example, the following statements create a copy of the first item in the collection and of the feature named Help_Files:
		`Set pFeat1 = m_ISWiProject.ISWiSuiteFeatures.Item(1)`
		`Set pFeat2 = m_ISWiProject.ISWiSuiteFeatures.Item("Help_Files")`
		The name of the feature is case-sensitive—for example, "Help_Files" and "Help_files" are two different features.
		Item is the default property for ISWiSuiteFeatures, which means that the following two lines of code are equivalent:
		`m_ISWiProject.ISWiSuiteFeatures.Item("Help_Files")`
		`m_ISWiProject.ISWiSuiteFeatures("Help_Files").`

Applies To

- ISWiProject

ISWiSuiteFiles Collection



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteFiles is a collection that contains all of the files in a folder of a package in an Advanced UI or Suite/Advanced UI project.

Members

Table 12-85 • ISWiSuiteFiles Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteFiles collection.
Item	Read-Only Property	Provide the index number or name of the file to retrieve the ISWiSuiteFile object.

Applies To

- ISWiSuiteFolder

ISWiSuiteFolders Collection



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteFolders is a collection that contains all of the subfolders in a folder of a package in an Advanced UI or Suite/Advanced UI project.

Members

Table 12-86 • ISWiSuiteFolders Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteFolders collection.
Item	Read-Only Property	Provide the index number or name of the subfolder to retrieve the ISWiSuiteFolder object.

Applies To

- ISWiSuiteFolder

ISWiSuitePackages Collection



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuitePackages is a collection that contains every package (as an ISWiSuitePackage object) in a Windows Installer transaction in your project.

Members

Table 12-87 • ISWiSuitePackages Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWISuitePackages collection.
Item	Read-Only Property	Provide the index number or name of the transaction to retrieve the ISWISuitePackage object.

Applies To

- ISWiProject

ISWiSuiteReleases Collection



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

ISWiSuiteReleases is a collection that contains all of the releases in your Advanced UI or Suite/Advanced UI project.

Members

Table 12-88 • ISWiSuiteReleases Collection Members

Name	Type	Description
Count	Read-Only Property	Use this property to return the total number of elements in the ISWiSuiteReleases collection.
Item	Read-Only Property	Provide the index number or name of the release to retrieve the ISWiSuiteRelease object.

Applies To

- ISWiProject

Object Reference for Expressions in Advanced UI and Suite/Advanced UI Projects



Project • *This information applies to the following project types:*

- *Advanced UI*
- *Suite/Advanced UI*



Edition • *The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

In various settings of Advanced UI and Suite/Advanced UI projects, you can embed object expressions to query target systems for information about a file, a registry entry, or other items. Object expressions use this convention:

```
[@Object(Parameters, ...).Property(Parameters, ...)]
```

Each object expression contains a reference to an object, which is a collection of object-specific properties. Objects and properties may contain parameters.

This section of the documentation explains each of the objects that are available for use in object expressions.

Table 12-1 • Objects That Are Available for Object Expressions

Object	Description
Feature Object	Check target systems for information about a feature in an Advanced UI or Suite/Advanced UI installation.
File Object	Check target systems for information about a file during an Advanced UI or Suite/Advanced UI installation.
Parcel Object	Check target systems for information about a package in an Advanced UI or Suite/Advanced UI installation.
Platform Object	Check target systems for operating system details during an Advanced UI or Suite/Advanced UI installation.
Registry Object	Check the registry on target systems during an Advanced UI or Suite/Advanced UI installation.

Feature Object

To create an object expression that checks target systems for information about a feature in an Advanced UI or Suite/Advanced UI installation, use the Feature object. The Feature object has support for properties such as DisplayName, ActionState, and InstalledState.

Syntax

```
[@Feature(FeatureName).ActionState]
[@Feature(FeatureName).AllowSelectionChange]
[@Feature(FeatureName).Cost]
[@Feature(FeatureName).Description]
[@Feature(FeatureName).DisplayName]
[@Feature(FeatureName).InstalledState]
[@Feature(FeatureName).Visible]
```

Parameters

One parameter is available for the Feature object.

Table 12-2 • Parameters for the Feature Object

Parameter	Description
FeatureName	Specify the name of the feature.

Properties

The following properties are available for the Feature object.

Table 12-3 • Properties for the Feature Object

Property	Description
ActionState	Retrieves the currently set action state of the feature. Valid values are: • **0**—No action will be taken for the feature. • **1**—The feature will be installed. • **2**—The feature will be removed.
AllowSelectionChange	Retrieves a Boolean value that indicates whether end users should be allowed to select or deselect the feature on the InstallationFeatures wizard page. Valid values are: • **true**—End users should be able to select or deselect the feature through the wizard interface. This is the default option. • **false**—End users should not be able to select or deselect the feature through the wizard interface. If this feature is visible on the InstallationFeatures wizard page, it appears in a read-only state (either selected or deselected, based on other factors, such as the feature's condition).
Cost	Retrieves how much space (in bytes) the feature requires on the target system if this value was specified in the Features view.
Description	Retrieves the localized description of the feature.

Table 12-3 • Properties for the Feature Object (cont.)

Property	Description
DisplayName	Retrieves the localized display name of the feature.
InstalledState	Retrieves the installed state of the feature as determined by the package detection states of the packages in the feature. Valid values are: • **0**—The feature is not currently installed. • **3**—The feature is installed.
Visible	Retrieves a Boolean value that indicates whether the feature should be visible on the InstallationFeatures wizard page during installation.

Examples

The following examples demonstrate use of the Feature object in object expressions.

Table 12-4 • Sample Feature Object Expressions

Expression	Description
[@Feature(FeatureABC).ActionState]	Determine whether the Advanced UI or Suite/Advanced UI is installing the FeatureABC feature.
[@Feature(FeatureABC).Cost]	Determine how much space (in bytes) the feature requires.

File Object

To create an object expression that checks target systems for information about a file during an Advanced UI or Suite/Advanced UI installation, use the File object. The File object has support for properties such as Date and Version.

Syntax

```
[@File(FilePath,64Bit).Content]
[@File(FilePath,64Bit).Date(Parameter)]
[@File(FilePath,64Bit).Version]
```

Parameters

The following parameters are available for the File object.

Table 12-5 • Parameters for the File Object

Parameter	Description
FilePath	Specify the path to the file. You can use property directory identifiers such as [ProgramFilesFolder].
64Bit	Use this optional Boolean parameter to indicate whether the file is in a 64-bit location, such as in the System32 folder on a 64-bit system. Valid options are: • **true**—Check the 64-bit location. • **false**—Check the 32-bit location. If you omit this property, or if or the target system is running a 32-version of Windows, the installation checks the 32-bit location.

Properties

The following properties are available for the File object.

Table 12-6 • Properties for the File Object

Property	Description
Content	If the file is a text file (for example, .txt, .htm, .xml, .config, .ini, or .sql), retrieves the contents of the file.
Date	Retrieves the date and time for the file. Optional parameters that are available for this property are: • **modified**—Retrieves the date and time at which the file was last modified. This is the default parameter. • **access**—Retrieves the date and time at which the file was last accessed. • **create**—Retrieves the date and time at which the file was created. If you omit a parameter for the Date property, the File object returns the last-modified date.
Version	Retrieves the version of the file. For an unversioned file, this property returns empty.

Examples

The following examples demonstrate use of the File object in object expressions

Table 12-7 • Sample File Object Expressions

Expression	Description
[@File([ProgramFilesFolder]MyCompany\MyProduct.exe,false).Date(modified)]	Obtain the last modified date and time for a file called **MyProduct.exe**. On 32-bit target systems, the path may be this: **C:\Program Files\MyCompany\MyProduct.exe** On 64-bit target systems, the path may be this: **C:\Program Files (x86)\MyCompany\MyProduct.exe**
[@File([SystemFolder]notepad.exe).Version]	Obtain the version number of **notepad.exe** in the System32 folder on 32-bit target systems or in the SysWOW64 folder on 64-bit target systems.
[@File([SystemFolder]notepad.exe,true).Version]	Obtain the version number of **notepad.exe** in the System32 folder on 32-bit and 64-bit target systems.

Parcel Object

To create an object expression that checks target systems for information about a package in an Advanced UI or Suite/Advanced UI installation, use the Parcel object. The Parcel object has support for properties such as Eligible, ActionState, and DetectedState.

Syntax

`[@Parcel(`*`PackageGUID`*`).ActionState]`

`[@Parcel(`*`PackageGUID`*`).DetectedState]`

`[@Parcel(`*`PackageGUID`*`).Eligible]`

`[@Parcel(`*`PackageGUID`*`).ExitCode]`

`[@Parcel(`*`PackageGUID`*`).Type]`

Parameters

One parameter is available for the Parcel object.

Table 12-8 • Parameters for the Parcel Object

Parameter	Description
PackageGUID	Specify the GUID of the package.

Properties

The following properties are available for the Parcel object.

Table 12-9 • Properties for the Parcel Object

Property	Description
ActionState	Retrieves the currently set action state of the package. This value may change, depending on when at run time the property is queried. Valid values are: • 1—The package will run its install operation. • 2—The package will run its modify operation. • 3—The package will run its repair operation. • 4—The package will run its remove operation. • 5—No action will be performed for the package.
DetectedState	Retrieves the installed state of the package. Valid values are: • 0—The package is not installed. • 1—The package is currently installed.
Eligible	Retrieves a Boolean value that indicates whether the target system meets the requirements that are necessary for the package to be run.
ExitCode	Retrieves the exit code for the package once its scheduled operation has completed. The value of this property is indeterminate before package operations are run.
Name	Retrieves the display name of the package.
Type	Retrieves the type of package. Valid values are: • 0—Executable package (.exe) • 1—Windows Installer package (.msi) • 2—Patch package (.msp) • 3—InstallScript package (.hdr) • 5—Sideloading UWP app package (.appx) • 6—Web Deploy package (.zip)

Examples

The following examples demonstrate use of the Parcel object in object expressions.

Table 12-10 • Sample Parcel Object Expressions

Expression	Description
[@Parcel(0F283EC0-E9D1-4D18-801D-0014F6CA96DF).DetectedState]	Determine whether the referenced package is installed.
[@Parcel(0F283EC0-E9D1-4D18-801D-0014F6CA96DF).ExitCode]	Obtain the exit code for the package once its scheduled operation has completed.

Platform Object

To create an object expression that checks target systems for operating system details during an Advanced UI or Suite/Advanced UI installation, use the Platform object. The Platform object has support for properties such as Architecture, FullVersion, and MajorVersion.

Syntax

```
[@Platform.Architecture]
[@Platform.BuildNumber]
[@Platform.CSDVersion]
[@Platform.FullVersion]
[@Platform.MajorVersion]
[@Platform.MinorVersion]
[@Platform.ProductType]
[@Platform.ServicePackVersion]
[@Platform.Value]
```

Parameters

The Platform object does not have any parameters.

Properties

The following properties are available for the Platform object.

Table 12-11 • Properties for the Parcel Object

Property	Description
Architecture	Retrieves the architecture (x86, x64, IA64, ARM, or Unknown) of the machine on which the Advanced UI or Suite/Advanced UI installation is running.

Table 12-11 • Properties for the Parcel Object (cont.)

Property	Description
BuildNumber	Retrieves the build number version of Windows on the machine on which the Advanced UI or Suite/Advanced UI installation is running. For a list of valid build numbers, see the dwBuildNumber member of the OSVERSIONINFOEX structure that is documented in the MSDN Library.
CSDVersion	Retrieves the CSD version of Windows on the machine on which the Advanced UI or Suite/Advanced UI installation is running. Service Pack 2 is an example of a CSD version.
FullVersion	Retrieves the full version number of Windows on the machine on which the Advanced UI or Suite/Advanced UI installation is running. The value is calculated as MajorVersion X 100 + MinorVersion. For example, the version number of Windows 8.1 is 6.3. The FullVersion value on a machine with Windows 8.1 is 603.
MajorVersion	Retrieves the major version number of Windows on the machine on which the Advanced UI or Suite/Advanced UI installation is running.
MinorVersion	Retrieves the minor version number of Windows on the machine on which the Advanced UI or Suite/Advanced UI installation is running.
ProductType	Retrieves the product type of the machine on which the Advanced UI or Suite/Advanced UI installation is running. Valid values are: • **1**—Workstation • **2**—Domain controller • **3**—Server
ServicePackVersion	Retrieves the service pack version of Windows on the machine on which the Advanced UI or Suite/Advanced UI installation is running.
Value	Retrieves the same value as the FullVersion property.

Examples

The following examples demonstrate use of the Platform object in object expressions.

Table 12-12 • Sample Platform Object Expressions

Expression	Description
[@Platform.FullVersion]	Obtain the version number of the target system.
[@Platform.Architecture]	Obtain the architecture of the target system.

Registry Object

To create an object expression that checks the registry on target systems during an Advanced UI or Suite/Advanced UI installation, use the Registry object.

Syntax

```
[@Registry(RegistryKeyPath,64Bit).KeyValue(DataValueName)]
```

```
[@Registry(RegistryKeyPath,64Bit).Value]
```

Parameters

The following parameters are available for the Registry object.

Table 12-13 • Parameters for the Registry Object

Parameter	Description
RegistryKeyPath	Specify the full path of the registry key that you want to check on target systems. Use the following abbreviations for the root: • HKCR • HKCU • HKLM
64Bit	Use this optional Boolean parameter to indicate whether to check the 64-bit view of the registry. Valid options are: • **true**—Check the 64-bit view of the registry. • **false**—Check the 32-bit view of the registry. If you omit this property, or if the target system is running a 32-version of Windows, the installation checks the 32-bit view of the registry.

Properties

The following properties are available for the Registry object.

Table 12-14 • Properties for the Registry Object

Property	Description
KeyValue	Retrieve the value data of the value that is passed as a parameter for the KeyValue property.
Value	Retrieve the value data of the registry key's default value.

Examples

The following examples demonstrate use of the Registry object in object expressions.

Table 12-15 • Sample Registry Object Expressions

Expression	Description
[@Registry(HKLM\Software\Microsoft\ Windows NT\CurrentVersion, false). KeyValue(RegisteredOwner)]	On 32-bit target systems, obtain the value data of the RegisteredOwner value for the following registry key: HKLM\Software\Microsoft\Windows NT\CurrentVersion On 64-bit target systems, obtain the value data of the RegisteredOwner value for the following registry key: HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion
[@Registry(HKLM\Software\[COMPANY]\ [PRODUCT], true).Value]	On 32-bit and 64-bit target systems, obtain the value data of the default value for the following registry key: HKLM\Software\My Company Name\My Product Name
[@Registry(HKCR\Software\My Company Name\My Product Name, false).KeyValue(MyFile[\[]1[\]].exe)]	On 32-bit target systems, obtain the value data of the MyFile[1].exe value for the following registry key: HKCU\Software\My Company Name\My Product Name On 64-bit target systems, obtain the value data of the MyFile[1].exe value for the following registry key: HKLM\Software\Wow6432Node\My Company Name\My Product Name To avoid treating the square brackets in the data value name as parameter parse indicators, the square brackets are escaped as shown in the sample expression.

InstallShield Custom Action Reference

This section describes each of the built-in InstallShield custom actions that are added automatically to InstallShield projects to support different functionality.



Windows Logo • *If you are applying for the Windows logo, the intended behavior of each custom action in your installation must be documented. This does not apply to custom action types 19 (displays an error, returns failure, and ends the installation), 35 (sets the installation directory), or 51 (sets a property). ISICE10 validates that each custom action is documented.*

_serial_verifyCA_isx

Decreases the value of the property **SERIALNUMVALRETRYLIMIT** by 1.

_serial_verifyCA_isx_helper

Decreases the value of the property **SERIALNUMVALRETRYLIMIT** by 1.

CheckForProductUpdates

Uses FlexNet Connect to check for product updates.

This custom action launches an executable file called **Agent.exe**, and it passes the following:

```
/au[ProductCode] /EndOfInstall
```

CheckForProductUpdatesOnReboot

Uses FlexNet Connect to check for product updates on reboot.

This custom action launches an executable file called **Agent.exe**, and it passes the following:

```
/au[ProductCode] /EndOfInstall /Reboot
```

DLLWrapCleanup

Standard DLL wrapper that cleans extracted data.

This is a Windows Installer DLL custom action. The name of the DLL file is **dllwrap.dll**, and its entry point is DLLWrapCleanup.

DLLWrapStartup

Standard DLL wrapper that extracts data that describes calls.

This is a DLL custom action. The name of the DLL file is **dllwrap.dll**, and its entry point is DLLWrapStartup.

ISChainPackageCommit

Removes extracted files from a multi-package transaction.

ISChainPackagePrepare

Prepares and extracts files for a multi-package transaction.

ISChainPackageRollback

Removes extracted files from a multi-package transaction.

ISComponentServiceCosting

Extracts information from the **ISComPlusApplication** table and saves it in a temporary file for COM+ applications.

This is a Windows Installer DLL custom action. The name of the DLL file is **iscomsrv.dll**, and its entry point is ISComponentServiceCosting.

ISComponentServiceFinalize

Commits COM+ applications during installation and uninstallation.

This is a Windows Installer DLL custom action. The name of the DLL file is **iscomsrv.dll**, and its entry point is ISComponentServiceFinalize.

ISComponentServiceInstall

Installs COM+ applications during an installation.

This is a DLL custom action. The name of the DLL file is **iscomsrv.dll**, and its entry point is ISComponentServiceInstall.

ISComponentServiceRollback

Rolls back COM+ applications during a failed installation or uninstallation.

This is a DLL custom action. The name of the DLL file is **iscomsrv.dll**, and its entry point is ISComponentServiceRollback.

ISComponentServiceUninstall

Removes COM+ applications during uninstallation.

This is a DLL custom action. The name of the DLL file is **iscomsrv.dll**, and its entry point is ISComponentServiceUninstall.

ISIISCleanup

Removes temporary files and registry entries for an IIS installation.

This is a DLL custom action. The name of the DLL file is **IISHelper.dll**, and its entry point is ISIISCleanup.

ISIISCosting

Creates a list of actions in a temporary file for an IIS installation. Sets the **CustomActionData** property for other IIS actions.

This is a DLL custom action. The name of the DLL file is **IISHelper.dll**, and its entry point is ISIISCosting.

ISIISInstall

Creates Web sites, applications, virtual directories, and other items for an IIS installation.

This is a DLL custom action. The name of the DLL file is **IISHelper.dll**, and its entry point is ISIISInstall.

ISIISRollback

Removes Web sites, applications, virtual directories, and other items during rollback for IIS.

This is a DLL custom action. The name of the DLL file is **IISHelper.dll** and its entry point is ISIISRollback.

ISIISUninstall

Removes Web sites, applications, virtual directories, and other items during uninstallation for IIS.

This is a DLL custom action. The name of the DLL file is **IISHelper.dll**, and its entry point is ISIISUninstall.

ISInstallPrerequisites

Launches the installation of prerequisites that are associated with features.

This is a DLL custom action. The name of the DLL file is **PrqLaunch.dll**, and its entry point is InstallPrerequisites.

ISJITCompileActionAtInstall

Precompiles .NET assemblies at installation time.

This custom action launches a Microsoft executable file called **ngen.exe**.

ISJITCompileActionAtUnInstall

Deletes a precompiled .NET assemblies at uninstallation time.

This custom action launches a Microsoft executable file called **ngen.exe**.

ISLockPermissionsCost

Sets the **CustomActionData** property for the ISLockPermissionsInstall action.

This is a DLL custom action. The name of the DLL file is **ISLockPermissions.dll**, and its entry point is ISLockPermissionsCostAction.

ISLockPermissionsInstall

Sets permissions when the product is installed.

This is a DLL custom action. The name of the DLL file is **ISLockPermissions.dll**, and its entry point is ISLockPermissionsInstallAction.

ISNetApiInstall

Creates users and groups from an .ini file.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetAPI.dll**, and its entry point is ISNetApiInstall.

ISNetApiRollback

Rolls back changes for users and groups.

This is a DLL custom action. The name of the DLL file is **ISNetAPI.dll**, and its entry point is ISNetApiRollback.

ISNetCreateIniForOneUser

Extracts operations to a temporary file for users and groups.

This is a DLL custom action called **ISNetAPI.dll**.

ISNetDeleteIniFile

Cleans up temporary files for users and groups.

This is a DLL custom action called **ISNetAPI.dll**.

ISNetGetGroups

Puts groups into a combo box.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetAPI.dll**, and its entry point is ISNetGetGroups.

ISNetGetServers

Puts a list of servers into a combo box.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetAPI.dll**, and its entry point is ISNetGetServers.

ISNetGetUsers

Puts a list of users into a combo box.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetAPI.dll**, and its entry point is ISNetGetUsers.

ISNetSetLogonName

Stores the user, group, and server properties that were entered in the LogonInformation dialogs.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetAPI.dll**, and its entry point is ISNetSetLogonName.

ISNetValidateLogonName

Verifies that a valid user name, server, and password combination were entered in the LogonInformation dialogs.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetAPI.dll**, and its entry point is ISNetValidateLogonName.

ISNetValidateNewUserInformation

Stores the user, group, and server properties that were entered in the LogonInformation dialogs.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetAPI.dll**, and its entry point is ISNetValidateNewUserInformation.

ISNetworkSharesCosting

Extracts information from the **ISNetworkShares** table and saves it in a temporary file for network sharing.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetworkShares.dll**, and its entry point is ISNetworkSharesCosting.

ISNetworkSharesFinalize

Cleans up temporary files for configuring network shares.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetworkShares.dll**, and its entry point is ISNetworkSharesFinalize.

ISNetworkSharesInstall

Configures network sharing for one or more folders.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetworkShares.dll**, and its entry point is ISNetworkSharesInstall.

ISNetworkSharesRollback

Rolls back network sharing for one or more folders.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetworkShares.dll**, and its entry point is ISNetworkSharesRollback.

ISNetworkSharesUninstall

Removes network sharing for one or more folders.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISNetworkShares.dll**, and its entry point is ISNetworkSharesUninstall.

ISPrint

Prints the contents of a ScrollableText control on a dialog.

This is a Windows Installer DLL custom action. The name of the DLL file is **SetAllUsers.dll**, and its entry point is PrintScrollableText.

ISQuickPatchFinalize

Cleans shared reference counts for a QuickPatch.

This is a Windows Installer DLL custom action. The name of the DLL file is **QuickPatchHelper.dll**, and its entry point is ISQuickPatchFinalize.

ISQuickPatchFixShortcut

Reinstalls shortcuts for a QuickPatch.

This is a Windows Installer DLL custom action. The name of the DLL file is **QuickPatchHelper.dll**, and its entry point is ISQuickPatchFixShortcut.

ISQuickPatchHelper

Applies clean feature states for a QuickPatch.

This is a Windows Installer DLL custom action. The name of the DLL file is **QuickPatchHelper.dll**, and its entry point is ISQuickPatchHelper.

ISQuickPatchInit

Cleans component and feature states for a QuickPatch.

This is a Windows Installer DLL custom action. The name of the DLL file is **QuickPatchHelper.dll**, and its entry point is ISQuickPatchInit.

ISQuickPatchInit9X

Cleans component and feature states for a QuickPatch.

This is a Windows Installer DLL custom action. The name of the DLL file is **QuickPatchHelper.dll**, and its entry point is ISQuickPatchInit9X.

ISQuickPatchInit9X2

Cleans component and feature states for a QuickPatch.

This is a Windows Installer DLL custom action. The name of the DLL file is **QuickPatchHelper.dll**, and its entry point is ISQuickPatchInit9X2.

ISRunSetupTypeAddLocalEvent

Runs the AddLocal events that are associated with the Next button on the SetupType dialog. This action needs to be called when the installation does not display the SetupType dialog.

This is a Windows Installer DLL custom action. The name of the DLL file is **ISXExpHlp.dll**, and its entry point is RunSetupTypeAddLocalEvent.

ISSearchReplaceCosting

Creates a temporary file that has a list of changes that need to be made in text files.

This is a DLL custom action. The name of the DLL file is **isschrpl.dll**, and its entry point is ISSearchReplaceCosting.

ISSearchReplaceFinalize

Cleans up temporary files for text file changes.

This is a DLL custom action. The name of the DLL file is **isschrpl.dll**, and its entry point is ISSearchReplaceFinalize.

ISSearchReplaceInstall

Applies text file changes when the product is installed.

This is a DLL custom action. The name of the DLL file is **isschrpl.dll**, and its entry point is ISSearchReplaceInstall.

ISSearchReplaceRollback

Rolls back text file changes and cleans up temporary files for text file changes.

This is a DLL custom action. The name of the DLL file is **isschrpl.dll**, and its entry point is ISSearchReplaceRollback.

ISSearchReplaceUninstall

Applies text file changes when the product is uninstalled.

This is a DLL custom action. The name of the DLL file is **isschrpl.dll**, and its entry point is ISSearchReplaceUninstall.

ISSelfRegisterCosting

Extracts operations to a temporary file for self-registration.

This is a Windows Installer DLL custom action. The name of the DLL file is **isregsvr.dll**, and its entry point is ISSelfRegisterCosting.

ISSelfRegisterFiles

Registers self-registering files.

This is a Windows Installer DLL custom action. The name of the DLL file is **isregsvr.dll**, and its entry point is ISSelfRegisterFiles.

ISSelfRegisterFinalize

Cleans up temporary file from self-registration.

This is a Windows Installer DLL custom action. The name of the DLL file is **isregsvr.dll**, and its entry point is ISSelfRegisterFinalize.

ISSetAllUsers

Sets **ALLUSERS** per upgrade or initial installation requirements.

This is a DLL custom action. The name of the DLL file is **SetAllUsers.dll**, and its entry point is SetAllUsers.

ISSetTARGETDIR

Sets TARGETDIR to **[INSTALLDIR]**.

This is a DLL custom action. The name of the DLL file is **SetAllUsers.dll**, and its entry point is SetTARGETDIR.

ISSetupFilesCleanup

Cleans up the temp directory of support files.

This is a DLL custom action. The name of the DLL file is **SFHelper.dll**, and its entry point is SFCleanupEx.

ISSetupFilesExtract

Extracts support files to a temp directory.

This is a DLL custom action. The name of the DLL file is **SFHelper.dll**, and its entry point is SFStartupEx.

ISSQLQueryDatabases

Finds database catalogs that are available on a specified database server.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLQueryDatabases.

ISSQLServerCosting

Extracts operations to a temporary file for SQL scripts.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerCosting.

ISSQLServerFilteredList

Finds available servers for SQL scripts.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerFilteredList.

ISSQLServerFinalize

Cleans up temporary files and the state for SQL scripts.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerFinalize.

ISSQLServerInitialize

Adds property names for SQL login password to **MsiHiddenProperties**.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerInitialize.

ISSQLServerInstall

Runs SQL scripts during an installation.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerInstall.

ISSQLServerList

Finds available servers.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerList.

ISSQLServerRemoveLoginInfo

Removes SQL database login credentials from the Windows registry during an uninstallation.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerRemoveLoginInfo.

ISSQLServerRollback

Runs SQL scripts during a failed installation or uninstallation.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerRollback.

ISSQLServerRollbackLoginInfo

During a failed installation or uninstallation, rolls back SQL database login credentials that are stored in the Windows registry.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerRollbackLoginInfo.

ISSQLServerUninstall

Runs SQL scripts during an uninstallation.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerUninstall.

ISSQLServerValidate

Tests server connections.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerValidate.

ISSQLServerWriteLoginInfo

Writes SQL database login credentials to the Windows registry during an installation.

This is a DLL custom action. The name of the DLL file is **issqlsrv.dll**, and its entry point is ISSQLServerWriteLoginInfo.

ISUnSelfRegisterFiles

Unregisters self-registering files.

This is a DLL custom action. The name of the DLL file is **isregsvr.dll**, and its entry point is ISUnSelfRegisterFiles.

ISVerifyScriptingRuntime

Verifies that an InstallScript MSI installation was launched using **Setup.exe**, and if not, it displays the message contained in the property STANDARD_USE_SETUPEXE. This custom action is included in all InstallScript MSI projects, and it is scheduled for the beginning of the User Interface sequence.



***Note** • The ISVerifyScriptingRuntime custom action is called in an InstallScript MSI installation in which the InstallScript UI style is the traditional style (which uses the InstallScript engine as an external UI handler). It is not called for the new style (which uses the InstallScript engine as an embedded UI handler).*

To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

ISXmlAppSearch

Performs a system search for XML: finds file, performs XPath queries, and stores the results in properties. Driven by the **ISXmlLocator** table.

This is a DLL custom action. The name of the DLL file is **isxmlcfg.dll**, and its entry point is ISXmlAppSearch.

ISXmlCosting

Extracts operations to a temporary file for XML file changes.

This is a DLL custom action. The name of the DLL file is **isxmlcfg.dll**, and its entry point is ISXmlCosting.

ISXmlFinalize

Cleans up temporary files for XML file changes.

This is a DLL custom action. The name of the DLL file is **isxmlcfg.dll**, and its entry point is ISXmlFinalize.

ISXmlInstall

Applies XML file changes.

This is a DLL custom action. The name of the DLL file is **isxmlcfg.dll**, and its entry point is ISXmlInstall.

ISXmlRollback

Rolls back XML file changes.

This is a DLL custom action. The name of the DLL file is **isxmlcfg.dll**, and its entry point is ISXmlRollback.

ISXmlUnInstall

Uninstalls XML file changes.

This is a DLL custom action. The name of the DLL file is **isxmlcfg.dll**, and its entry point is ISXmlUnInstall.

LaunchProgramFileFromSetupCompleteSuccess

Launches an executable file at the end of an installation.

This is a Windows Installer DLL custom action. The name of the DLL file is **SerialNumCAHelper.dll**, and its entry point is LaunchProgram.

LaunchReadmeFileFromSetupCompleteSuccess

Launches a readme file at the end of an installation.

This is a Windows Installer DLL custom action. The name of the DLL file is **SerialNumCAHelper.dll**, and its entry point is LaunchReadMe.

setAllUsersProfile2K

Initializes the ALLUSERSPROFILE directory identifier for Windows 2000 or later.

SetAllUsersProfileNT

Initializes the ALLUSERSPROFILE directory identifier for Windows NT 4.

SetARPINSTALLLOCATION

Sets the value of the **ARPINSTALLLOCATION** property to the fully qualified path for the application's primary folder.

setUserProfileNT

Initializes the USERPROFILE directory indentifier.

ShowMsiLog

Displays the Windows Installer log file in Notepad if the end user selects the **Show the Windows Installer log** check box in the SetupCompleteSuccess, SetupCompleteError, or SetupInterrupted dialogs and then clicks Finish. This works only with Windows Installer 4.0 and later.

WiseScriptFinalize

Cleans up temporary files for a WiseScript custom action.

This is a a DLL custom action. The name of the DLL file is **WiseScriptBridge.dll**, and its entry point is WiseScriptFinalize.

WiseScriptInitialize

Extracts required files for a WiseScript custom action.

This is a a DLL custom action. The name of the DLL file is **WiseScriptBridge.dll**, and its entry point is WiseScriptInitialize.

WiseScriptRollback

Cleans up temporary files for a WiseScript custom action.

This is a a DLL custom action. The name of the DLL file is **WiseScriptBridge.dll**, and its entry point is WiseScriptFinalize.

Command-Line Tools

In addition to its graphical user interface, InstallShield provides several command-line programs that you can use during build time, for instance, as part of a batch process, or during run time to customize the behavior of your installation.

Authoring-Time Programs

Table 12-1 • Descriptions of Authoring-Time Programs

Task	Program
Scan an IIS Web site and record its settings so that you can import them into an InstallShield project	iisscan.exe

Build-Time Programs

Table 12-2 • Descriptions of Build-Time Programs

Task	Program
Compile an InstallScript file (.rul)	Compile.exe
Build a release; if applicable, also compile the InstallScript	ISCmdBld.exe
Build a release (available for legacy InstallScript projects)	ISBuild.exe
Sign an already-built InstallScript release	iSign.exe
Build a release (InstallScript self-extracting executable files)	ReleasePackager.exe
Modify a Setup.ini file that is embedded in a Basic MSI or InstallScript MSI Setup.exe file	SetupIni.exe

Run-Time Programs

Table 12-3 • Descriptions of Run-Time Programs

Task	Program
Run a Setup.exe installation and pass command-line parameters (Basic MSI, InstallScript, and InstallScript MSI projects)	Setup.exe
Run a Setup.exe installation and pass command-line parameters (Advanced UI and Suite/Advanced UI projects)	Setup.exe
Run an installation (InstallScript projects)	Setup.exe (InstallScript projects)
Run an .msi package	MsiExec.exe
Run a patch package (Windows Installer-based projects) and pass command-line parameters	Update.exe

Compile.exe

InstallShield includes a command-line compiler that you can invoke from the Command Prompt window or from a DOS batch file. This program, called **Compile.exe**, is in the following folder:

*InstallShield Program Files Folder***System**



Note *• When you build a release from the command line using* **ISCmdBld.exe***, the build engine automatically compiles your script; therefore, you do not need to use* **Compile.exe** *directly unless you want to use compiler options that are different from those specified for the project in InstallShield. For more information, see ISCmdBld.exe.*

Syntax

```
Compile script_file [iswi_obl_file] [isrt_obl_file] [ifx_obl_file] switches
```

You can pass any or all of your command-line parameters to **Compile.exe** in the form of a command file. To specify a command file to **Compile.exe**, use the following syntax:

```
Compile @command_file
```

In this example, *command_file* is the name of a text file, including any extension. You can specify an absolute or relative path with the file name.

You can specify all or part of the command line in a command file. You can use more than one command file in a Compile command. **Compile.exe** accepts the command file's input as if it were specified in that location on the command line.

In the command file, each parameter must begin and end on the same line—you cannot use the backslash (\) to combine a parameter across two lines. In the command file, arguments can be separated by spaces or tabs (as on the command line) and by newline (\n) characters.

Parameters

The following parameters are available for use with the compiler.

Table 12-4 • Command-Line Parameters for Compile.exe

Parameter	Description
script_file	Specifies the name of the setup script. Note the following: • The file name can include a drive designation and an absolute or relative path. If the file name is unqualified, the compiler will search for the script in the current directory of the current drive. • If the specified setup script uses #include directives to include other files in the setup, and if the file names specified by #include statements do not include a path, you must specify the location of those files with the -i switch unless they reside in the current directory. Note that included files may themselves specify #include directives; be sure the -i switch specifies a search path that can be used to find all included source files. • Long path and file names must be enclosed in double quotation marks. • By default, the setup script created as part of a new InstallShield project is named setup.rul.

Table 12-4 • Command-Line Parameters for Compile.exe (cont.)

Parameter	Description
isrt_obl_file	Specifies the name of the library file **Isrt.obl**. It is not necessary to include the full path for the library file, as long as the path is identified with the `-libpath` switch. **Isrt.obl** is in the following directory: *`InstallShield Program Files Folder`***\Script\Isrt\Lib** This parameter is required unless you are using the `-c` or `-l` switch.
ifx_obl_file	 ***Project*** • *This parameter applies to InstallScript projects.* Specifies the name of the library file **Ifx.obl**. It is not necessary to include the full path for the library file, as long as the path is identified with the `-libpath` switch. **Ifx.obl** is in the following directory: *`InstallShield Program Files Folder`***\Script\Ifx\Lib** This parameter is required unless you are using the `-c` or `-l` switch.
iswi_obl_file	 ***Project*** • *This parameter applies to the following project types:* • *Basic MSI* • *InstallScript MSI* Specifies the name of the library file **Iswi.obl**. It is not necessary to include the full path for the library file, as long as the path is identified with the `-libpath` switch. **Iswi.obl** is in the following directory: *`InstallShield Program Files Folder`***\Script\Iswi\Lib** This parameter is required unless you are using the `-c` or `-l` switch.
ifxobject_obl_file	 ***Project*** • *This parameter applies to InstallScript Object projects.* Specifies the name of the library file **Ifxobject.obl**. It is not necessary to include the full path for the library file, as long as the path is identified with the `-libpath` switch. **Ifxobject.obl** is in the following directory: *`InstallShield Program Files Folder`*\Script\Ifx\Lib This parameter is required unless you are using the `-c` or `-l` switch.

Switches

You can pass the following switches to the compiler. If you are not specifying the full path to the script files and library files, you must specify the location of the folders that contain those files by using the `-i` and `-libpath` switches (as noted in the `-i` and `-libpath` descriptions below). All of the other switches are optional.

Table 12-5 • Switches for Compile.exe

Switch	Description
/d or -d<variable name=preprocessor define>	Provides a preprocessor definition that is applicable to your InstallScript. Following is an example: `Compile.exe -dVARIABLENAME=Value` The variable name must be a valid InstallScript identifier; the value must be a constant. No space is permitted between the switch and the expression or within the expression. If you reference the preprocessor in InstallScript, use either of the following formats: • #ifdef VARIABLENAME • #if VARIABLENAME=Value That is, for #ifdef statements, you can use just the value name. For #if statements, use a name=value pair. This parameter is optional. ***Note*** *• It is recommended that you use this command-line parameter only if a preprocessor is not defined in either of the following areas in InstallShield:* • On the Compile/Link tab of the Settings dialog box, which is displayed when you click Settings on the Build menu • For Basic MSI and InstallScript MSI projects: In the Preprocessor Defines setting for the product configuration in the Releases view *If you define a preprocessor in either of these areas in InstallShield and then you use the* `-d` *command-line parameter to define the same preprocessor,* **Compile.exe** *displays a cannot-define-symbol message to inform you that the preprocessor cannot be redefined through an InstallShield project or product configuration setting.*
/e or -e	Specifies the maximum number of error messages. The default value is 50. When the maximum number of error messages has been generated, compilation stops.
/g or -g	Specifies that a debugging information file should be produced. This file is given the name of the installation and extension .dbg.

Table 12-5 • Switches for Compile.exe (cont.)

Switch	Description
/gi or -gi	Specifies that debugging information should be included in the compiled script file, so a debugging information file is not needed. This option makes the compiled script larger and the installation slower, and makes it easier for others to reverse engineer your code, so it should typically not be used when creating your final installation for distribution to end users.
/i or -i<search path>	Specifies a search path—a list of paths, each separated by a semicolon—that identifies the directories to search for source files that have been included in the main installation's InstallScript code through #include statements. Long path names must be enclosed in double quotation marks.  ***Project*** • *For InstallScript and InstallScript Object projects, you must specify the full paths for the following folders:* • *`InstallShield Program Files Folder`***\Script\Ifx\Include** • *`InstallShield Program Files Folder`***\Script\Isrt\Include** *For Basic MSI and InstallScript MSI projects, you must specify the full paths for the following folders:* • *`InstallShield Program Files Folder`***\Script\Iswi\Include** • *`InstallShield Program Files Folder`***\Script\Isrt\Include** *The aforementioned directories contain the script header files for built-in InstallScript functionality. If you do not specify these folders, compiler errors occur.*

Table 12-5 • Switches for Compile.exe (cont.)

Switch	Description
/libpath or -libpath<path>	Specifies a single path that identifies a directory to search for libraries such as **Ifx.obl**, **Isrt.obl**, or a custom library file. To specify multiple directories, use the `-libpath` switch multiple times. Long path names must be enclosed in double quotation marks.  ***Project*** *• For InstallScript and InstallScript Object projects, you must specify the full paths for the following folders (unless you have used the full path with the library file names):* • *`InstallShield Program Files Folder`***\Script\Ifx\Lib** • *`InstallShield Program Files Folder`***\Script\Isrt\Lib** *For Basic MSI and InstallScript MSI projects, you must specify the full paths for the following folders (unless you have used the full path with the library file names):* • *`InstallShield Program Files Folder`***\Script\Iswi\Lib** • *`InstallShield Program Files Folder`***\Script\Isrt\Lib** *The aforementioned directories contain the libraries that define the built-in InstallScript functions. If you do not specify these folders or the full paths with the library names, compiler errors occur.*
/o or -o<compiled script file name>	Specifies the file name to assign to the compiled script. Note the following: • The file name can include a drive designation and an absolute or relative path. If a path is not specified, the compiler will store the compiled script in the current directory. • If this parameter is not specified, the compiled script is given the name of the script file and the extension .inx and is stored in the current directory. • Long path and file names must be enclosed in double quotation marks.
/q or -q	Suppresses the output of the copyright message and version information.
/v or -v	Sets the warning level to one of the following values: • **0**—Displays no warning messages. • **1**—Displays any system warning message that InstallShield is unable to handle. • **2**—Displays Level 1 messages, plus a message if string length exceeds the limit. • **3**—Displays all warning messages. This is the default setting.
/w or -w	Specifies the maximum number of warning messages. The default value is 50. When the maximum number of warning messages has been generated, compilation stops.

Additional Information

- When specifying file names, you can use either short or long file names; however, long file names must be enclosed within double quotation marks.
- If you specify a relative path to a file name (for example **..\..\My Functions**), that path must be relative to the current directory of the current drive at the time that you invoke the compiler.

Examples

Sample Command Line for an InstallScript Project

The following command line for an InstallScript project compiles the **Setup.rul** script file that is located in the **C:\InstallShield 2016 Projects\My InstallScript Project\script files** folder:

```
"C:\Program Files\InstallShield\2016\System\Compile.exe"
"C:\InstallShield 2016 Projects\My InstallScript Project\script files\setup.rul"
ifx.obl
isrt.obl
-libpath"C:\Program Files\InstallShield\2016\Script\Ifx\Lib"
-libpath"C:\Program Files\InstallShield\2016\Script\Isrt\Lib"
-i"C:\Program Files\InstallShield\2016\Script\Ifx\Include"
-i"C:\Program Files\InstallShield\2016\Script\Isrt\Include"
-i"C:\InstallShield 2016 Projects\My InstallScript Project\script files"
```

Sample Command Line for an InstallScript MSI or Basic MSI Project

The following command line for an InstallScript MSI or Basic MSI project compiles the **Setup.rul** script file that is located in the **C:\InstallShield 2016 Projects\My New Project\script files** folder:

```
"C:\Program Files\InstallShield\2016\System\Compile.exe"
"C:\InstallShield 2016 Projects\My New Project\script files\setup.rul"
ISWI.obl
isrt.obl
-libpath"C:\Program Files\InstallShield\2016\Script\Iswi\Lib"
-libpath"C:\Program Files\InstallShield\2016\Script\Isrt\Lib"
-i"C:\Program Files\InstallShield\2016\Script\Ifx\Include"
-i"C:\Program Files\InstallShield\2016\Script\Isrt\Include"
-i"C:\InstallShield 2016 Projects\My New Project\script files"
```

ISCmdBld.exe

You can build a release from the command line using **ISCmdBld.exe** for Windows Installer–based projects and for InstallScript projects. If your project includes InstallScript, **ISCmdBld.exe** also compiles it before building the release.

Syntax

The following statement illustrates running **ISCmdBld.exe** to build the release **Othello Beta**:

```
ISCmdBld.exe -p "C:\InstallShield 2016 Projects\My Othello Project\Othello.ism" -r "Othello Beta" -c
    COMP -a "Build 245"
```

The first parameter in the example above, starting with `-p`, is the path to the .ism file that you would like to build. Next, `-r Othello Beta` is the name of the release. The parameter `-c COMP` specifies that you would like your package to be compressed into one file. Finally, `-a "Build 245"` points to the specific product configuration.

If a command-line build completes without any errors, InstallShield sets the environment variable ERRORLEVEL to 0. If an error occurs during a command-line build, ERRORLEVEL is set to 1. If ERRORLEVEL is set to any other value, this typically indicates an invalid parameter was passed to **ISCmdBld.exe**, and the specific cause of the error is displayed in the Command Prompt window in which **ISCmdBld.exe** was running.

Command-Line Parameters



Project • *Some of the* **ISCmdBld.exe** *command-line parameters are applicable to only certain project types.*

Table 12-6 • ISCmdBld.exe Command-Line Parameters

Parameter	Project Type	Description
-a <product configuration>	Basic MSI, InstallScript MSI, Merge Module	This parameter specifies the product configuration for the release. If it does not exist, it is created. Although this parameter is not required, you should include it if you are including the parameter for the release name.
-b <build location>	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	The fully qualified path to the folder where you want the output folders and files to be placed. UNC paths are acceptable. The built installation's files will be located in the Disk Images\Disk1 subfolder of the location that you specify. This parameter is optional. If it is left unspecified, the build will place the build package and files in the directory specified in the File Locations tab of the Options dialog box. Enclose long file names in quotation marks.
-c <release configuration>	Basic MSI, InstallScript MSI	This parameter enables you to specify whether your release is compressed into one file or remains uncompressed in multiple files. The valid arguments for this parameter are COMP and UNCOMP. To specify that your release be compressed into one file, use the COMP argument. If you do not want your release compressed, use the UNCOMP argument. If the parameter is omitted for a release that already exists, the configuration is based on what was specified in the InstallShield interface. If the parameter is omitted for a new release, the files remain uncompressed.

Table 12-6 • ISCmdBld.exe Command-Line Parameters (cont.)

Parameter	Project Type	Description
-d <variable name=preprocessor define>	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object	Use this parameter to provide a preprocessor definition that is applicable to your InstallScript. To specify more than one preprocessor definition, separate each with a comma, as in the following example: `ISCmdBld.exe -d VARIABLENAME1=Value1,VARIABLENAME2=Value2` If you reference the preprocessor in InstallScript, use either of the following formats: • #ifdef VARIABLENAME • #if VARIABLENAME=Value That is, for #ifdef statements, you can use just the value name. For #if statements, use a name=value pair. This parameter is optional.  ***Note*** • *It is recommended that you use this command-line parameter only if a preprocessor is not defined in either of the following areas in InstallShield:* • On the Compile/Link tab of the Settings dialog box, which is displayed when you click Settings on the Build menu • For Basic MSI and InstallScript MSI projects: In the Preprocessor Defines setting for the product configuration in the Releases view *If you define a preprocessor in either of these areas in InstallShield and then you use the* `-d` *command-line parameter to define the same preprocessor,* **ISCmdBld.exe** *displays a cannot-define-symbol message to inform you that the preprocessor cannot be redefined through an InstallShield project or product configuration setting.*
-e <Y/N>	Basic MSI, InstallScript MSI, Merge Module	For installation projects, this parameter specifies whether you want to create a **Setup.exe** file along with your installation. Specify `-e Y` to build **Setup.exe** or `-e N` to just create an installation. For merge module projects, the meaning of `-e` is different from installation projects. Specifying `-e n` will cause the merge module to be built and then copied to the merge modules folder. Specifying `-e Y` will cause the merge module to only be built but not copied to the merge modules folder.

Table 12-6 • ISCmdBld.exe Command-Line Parameters (cont.)

Parameter	Project Type	Description
-f <release flags>	Advanced UI, Basic MSI, InstallScript MSI, Suite/Advanced UI	Use this parameter to specify any release flags that you would like to include in your release. Separate multiple flags with commas.
-g <minimum target MSI version>	Basic MSI, InstallScript MSI, Merge Module	Use this parameter to specify the minimum version of Windows Installer that the installation requires on the target machine —for example, `2.0.2600.0`. This parameter is optional. The default is the latest version of Windows Installer that the InstallShield interface supports.
-h	Basic MSI, InstallScript MSI	To skip the upgrade validators at the end of the build, use this parameter.
-i <.ini file path>	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Instead of passing all of your parameters on the command line, you can include them all in an initialization (.ini) file and call that file from the command line. For more information, see Passing Command-Line Build Parameters in an .ini File. Absolute and relative paths are acceptable. Enclose long file names in quotation marks.
-j <minimum target Microsoft .NET Framework version>	Basic MSI, InstallScript MSI	This parameter specifies the minimum version of the .NET Framework that the installation requires on the target machine—for example, 1.0.3705.2. This parameter is optional. The default is the latest version of the .NET Framework that the InstallShield interface supports.
-l <path variable>="new path"	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/Advanced UI	Use this parameter to override a path variable that has been specified in the Path Variables view. You can specify this parameter multiple times, once for each path variable override. For example: `ISCmdBld.exe -l VariableName="C:\Path" -l VariableName2="C:\Path2"` To reference a file in a sibling project that is within the same Visual Studio solution folder, use the predefined path variable called VSSolutionFolder. For more information, see Using the VSSolutionFolder Path Variable with Visual Studio Solutions.

Table 12-6 • ISCmdBld.exe Command-Line Parameters (cont.)

Parameter	Project Type	Description
-m <.cub file name>	Basic MSI, InstallScript MSI, Merge Module	Use this parameter if you want to validate the installation or merge module package after it is built. Pass the path of the .cub file name with this parameter. Absolute and relative paths are acceptable. To use multiple .cub files, separate each path with a semicolon (;). Enclose long file names in quotation marks. For example, in the following command line, the `-m` parameter indicates that validation should be performed with the InstallShield Validation Suite for Windows 7 (**ISWin7.cub**) and the Full MSI Validation Suite (**darice.cub**): `ISCmdBld.exe -m "..\Support\Validation\ISWin7.cub;..\Support\Validation\darice.cub"`

-n	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Specify this parameter if you do not want **Setup.rul** to be compiled as part of the build process.
		
		Note that if you are building from the command line and specifying an .ini file with the build settings, you can specify the following in the `[Project]` section of the .ini file, instead of adding the `-n` command line build flag:
		
		`[Project]`
		
		`CompileScript=No`
		
		Like the command-line build flag, this CompileScript keyword helps you specify whether **Setup.rul** should be compiled as part of the build process.
		
		
		
		Note • *The* `-n` *parameter cannot be used with the* `-q3` *parameter.*

-o <merge module search path>	Basic MSI, InstallScript, InstallScript MSI	This parameter specifies one or more comma-delimited folders that contain the merge module files (.msm) that are referenced by your project. InstallShield provides additional ways for specifying the folders that contain merge modules. For more information, see Specifying the Directories that Contain Merge Modules.

Table 12-6 • ISCmdBld.exe Command-Line Parameters (cont.)

Parameter	Project Type	Description
-p <project location>	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	Pass the path to the project (.ism) file. This path can be fully qualified, relative, or just the file name. If only the project file name is passed, the file is retrieved relative to the current working directory. For example: `ISCmdBld.exe -p "C:\InstallShield 2016 Projects\MyProject1\MyProject1.ism"` UNC paths are also acceptable. Enclose long file names in quotation marks. This is the only required parameter.
-patch_config <patch configuration>	Basic MSI, InstallScript MSI	Use this parameter to build a standard patch. Pass the name of the patch configuration in the Patch Design view that you want to build. For example, the following command line builds the patch configuration called **Version 1.2** in the **MyProject1.ism** project: `ISCmdBld.exe –p "C:\InstallShield 2016 Projects\MyProject1\MyProject1.ism" -patch_config “Version 1.2”`
-prqpath <InstallShield prerequisite search path>	Basic MSI, InstallScript, InstallScript MSI	This parameter specifies one or more comma-delimited folders that contain the InstallShield prerequisite files (.prq) that are referenced by your project. InstallShield provides additional ways for specifying the folders that contain InstallShield prerequisite files. For more information, see Specifying the Directories that Contain InstallShield Prerequisites.
-q1	Basic MSI, InstallScript MSI, Merge Module	Builds only the Windows Installer tables for your release. If you have not built this installation already, a new .msi file is created, but no files are added to your installation. If you have built your installation already, the .msi file is updated when all the tables are built, but, again no files are transferred. Ideally, this option is to be used to test the user interface of your installation.  ***Note*** • *The* `-q1` *parameter cannot be used with the* `-q2` *or* `-q3` *parameters.*

Table 12-6 • ISCmdBld.exe Command-Line Parameters (cont.)

Parameter	Project Type	Description
-q2	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	For Basic MSI, InstallScript MSI, and Merge Module projects: Builds the Windows Installer tables and refreshes files. This option rebuilds your .msi file and updates the **Files** table, including any new or changed files in your installation. Changed files are updated only if the size or time stamp differs from the copy already included in the build. References to deleted files are removed from the installation, but the file remains in the build location. This type of build can be run only after a complete build has been performed, and it works only when the media is an uncompressed network image. For InstallScript and InstallScript Object projects: Rebuilds only those portions of the release that have changed since the last build. If this parameter is not used, the entire file media library is rebuilt.  ***Note*** • *The* `-q2` *parameter cannot be used with the* `-q1` *or* `-q3` *parameters.*
-q3	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	Compiles only **Setup.rul**. For Basic MSI projects, this parameter also streams **ISSetup.dll** into the **Binary** table of the .msi package, if one was previously built.  ***Note*** • *The* `-q3` *parameter cannot be used with the* `-q2` *or* `-q3`*, or* `-n` *parameters.*  ***Note*** • *This parameter does not upgrade the project. If your project was created with InstallShield Developer 8 or earlier, you should upgrade it to the latest version of InstallShield before using* `-q3`*. For example, use* `-u` *to upgrade.*
-r <release name>	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	The release name as specified in the Releases view. You can use an existing release name or create a new one. Although this parameter is not required, you should include it if you are including `-a`, the parameter for product configurations.

Table 12-6 • ISCmdBld.exe Command-Line Parameters (cont.)

Parameter	Project Type	Description
-s	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This parameter enables you to build your release in silent mode. Silent builds are useful if you want to run the build without displaying any errors or warning messages. This parameter is optional.
-t <Microsoft .NET Framework path>	Basic MSI, InstallScript MSI	**Regasm.exe** and **InstallUtilLib.dll** are utilities that are included with each version of the .NET Framework. This parameter specifies the path for the directory that contains the installed 32-bit version of these files that you want to use at build time for releases that include .NET installer classes and COM interop. This is not the path to .NET Framework redistributable files.  ***Note*** • *If you are using InstallShield on a 64-bit system,* **ISCmdBld.exe** *determines the location of the corresponding 64-bit version of the .NET Framework based on the path that you specify for this parameter, and* **ISCmdBld.exe** *uses the 64-bit location of* **Regasm.exe** *and* **InstallUtilLib.dll** *when appropriate.*
-u	Basic MSI	This parameter enables you to upgrade—but not build—your release. You can use this parameter to upgrade an installation project that you created using InstallShield—Windows Installer Edition version 2.03 or earlier.
-v	Advanced UI, Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module, Suite/ Advanced UI	This parameter enables you to create a verbose build log file that may be useful if Flexera Software Support is helping you troubleshoot an issue with your installation.
-w	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	This parameter treats warnings that occur during the build process as errors. Each warning increments the error count by one.

Table 12-6 • ISCmdBld.exe Command-Line Parameters (cont.)

Parameter	Project Type	Description
-x	Basic MSI, InstallScript, InstallScript MSI, InstallScript Object, Merge Module	If you want the build to stop when it encounters an error, use the `-x` parameter. If you want the build to stop when it encounters a warning, use this parameter in conjunction with the `-w` parameter.
-y <product version>	Advanced UI, Basic MSI, InstallScript MSI, Merge Module, Suite/Advanced UI	This parameter enables you to specify a product version from the command line. This is especially helpful if you want to increment the build version (the third field) of the product version. For example, to set the product version to 1.0.5: `ISCmdBld.exe -y "1.0.5"` For information on valid product version numbers, see Specifying the Product Version. This parameter is optional.

-z <property name=property value>	Advanced UI, Basic MSI, InstallScript MSI, Merge Module, Suite/Advanced UI	In Basic MSI, InstallScript MSI, and Merge Module projects, this parameter enables you to override the value of a Windows Installer property. In Advanced UI and Suite/Advanced UI projects, this parameter enables you to override the value of an Advanced UI or Suite/Advanced UI property.
		For example, to set the value of the property called **PropertyName** to **PropertyValue**, use the following command line:
		`IsCmdBld.exe -z PropertyName=PropertyValue`
		If you want to set the value to a string that includes one or more spaces, use quotation marks; for example, to set the value of the property called **PropertyName** to **My Property Value**, use the following command line:
		`IsCmdBld.exe -z "PropertyName=My Property Value"`
		If you specify a property that does not exist in the **Property** table, InstallShield creates the property in the installation that it builds.
		If you specify both the `-y` parameter to override the product version, and the `-z` parameter to override the **ProductVersion** property, InstallShield uses the value specified for the `-y` parameter.
		This parameter is optional.

ISBuild.exe



Project • *This information applies to InstallScript projects.*

ISBuild.exe is a command-line tool that is available for legacy InstallScript projects. The following parameters are associated with **ISBuild.exe**:

Table 12-7 • Command-Line Parameters for ISBuld.exe

Parameter	Description
-p ‹project file›	This specifies the fully qualified path and file name of the project file. By default, the project file is in the following location: **C:\InstallShield 2016 Projects***Project Name******Project File* This parameter should be enclosed in quotation marks.
-m ‹media name›	This specifies the media name for the media that will be built. This parameter should be enclosed in quotation marks.
-b ‹build location›	This specifies the fully qualified path to the folder where you want the output folders and files to be placed. This parameter is optional.
-s	This runs **ISBuild.exe** in silent mode. This parameter is optional.
-r	This indicates that the media should refresh the build. The default is a complete rebuild. This parameter is optional.

IISscan.exe

Edition • *The ability to import IIS data into an InstallShield project is available only in the Premier edition of InstallShield.*

The IIS scanner (**IISscan.exe**) is a tool that scans an IIS Web site and records the values of the settings that you can configure in the Internet Information Services view in InstallShield. **IISscan.exe** creates an XML file that contains all of the values. You can use this XML file to import the values into the Internet Information Services view. Then you can modify the settings as needed.

When you use **IISscan.exe** to scan a Web site, you must place the **IISscan.exe** file on the machine that contains the IIS Web site.

Note • **IISscan.exe** *requires administrative privileges. Therefore, you may need to launch it from an elevated Command Prompt window.*

IISscan.exe is installed in the following location:

*InstallShield Program Files Folder***System**

Syntax

```
iisscan.exe -website "Site Name" -outfile "C:\PathToFile\FileName.xml"
```

Parameters

Table 12-8 • Command-Line Parameters for IISscan.exe

Parameter	Description
-website	Specify the name of the Web site on the IIS server that you want to scan.
-outfile	Specify the full path and name of the XML file that you want the scanner to create. This parameter is optional. If you do not pass this parameter, the IIS scanner records the IIS settings in a file called iisscan.xml in the same directory that contains the IIS scanner.



Tip • *InstallShield lets you configure filters if you want to prevent certain IIS data—such as Web sites, applications, virtual directories, application pool, or any of their settings—from ever being imported into the Internet Information Services view. To learn more, see Filtering IIS Data When Importing a Web Site and Its Settings into an InstallShield Project.*

iSign.exe



Project • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript Object*

Use the command-line tool **iSign.exe** to digitally sign a release of an InstallScript project from the command line after you have built the release.

iSign.exe is in the following directory:

*InstallShield Program Files Folder***System**

iSign.exe lets you choose between the following options:

- You can specify the .pfx certificate file on your machine that you want to use for signing.
- You can reference a certificate store that contains the certificate that you want to use for signing.

Syntax

```
iSign.exe [options] PathToData1.hdrFile
```

For example, for signing with a .pfx file:

```
iSign.exe -pfx "C:\Temp\MyFile.pfx" -p "Password" "C:\InstallShield 2016\MyProject\Media\Release 1\Disk
    Images\Disk1\data1.hdr"
```

For signing with a certificate in a certificate store:

```
iSign.exe -store "Root" -sl "Machine" ss "My Certificate" "C:\InstallShield 2016\MyProject\Media\Release
    1\Disk Images\Disk1\data1.hdr"
```

Command-Line Parameters

Following is a list of the parameters that you can use with **iSign.exe**. Note that some are available only if you are using a .pfx file, and some are available only if you are using a certificate in a certificate store.

Table 12-9 • Command-Line Parameters for iSign.exe

Parameter	Signing Method	Description
-pfx <path to .pfx file>	.pfx file	Specify the fully qualified path to the .pfx file that contains the certificate that you want to use to sign your release.
-p <password>	.pfx file	If the .pfx file that you are using has a password, specify it with this parameter. If your certificate is protected by a password but you do not enter it for this parameter, signing with a .pfx file fails.  ***Note*** • *This parameter is not applicable if the certificate that you are using for signing is in a certificate store. Note that if the certificate was imported into the store with password protection, you will be prompted to enter the password while signing with* **iSign.exe**.
-store <store name>	certificate in certificate store	Specify the name of the certificate store that contains the certificate that you want to use. Available options are: • My • Root • Trust • CA
-sl <store location>	certificate in certificate store	Specify the location of the certificate store that contains the certificate that you want to use. Available options are: • User • Machine
-ss <subject>	certificate in certificate store	Specify the subject of the certificate that you want to use. This parameter is applicable if the certificate is in a certificate store.

ReleasePackager.exe



Project • *This information applies to InstallScript projects.*

Use **ReleasePackager.exe** to build a self-extracting executable file from the command line. This can be useful if you are building from a batch file.

ReleasePackager.exe is installed in the following location:

*InstallShield Program Files Folder***System**

Syntax

```
ReleasePackager.exe "disk_images_folder" "package_file" ["icon_file" [icon_index]]
```

Parameters

Table 12-10 • Command-Line Parameters for ReleasePackager.exe

Parameter	Description
disk_images_folder	This optional parameter specifies the folder that contains the disk images to be incorporated in the self-extracting executable file. A typical value is: **C:\InstallShield 2016 Projects\ProjectName\Media\Releasename\Disk Images**
package_file	This parameter specifies the fully qualified name of the self-extracting executable file. A typical value is: **C:\InstallShield 2016 Projects\Projectname\Media\Releasename\Package\Packagename.exe**
icon_file	This optional parameter specifies the fully qualified name of the file that contains the icon you want to use for your self-extracting executable file.
icon_index	This optional parameter specifies the numeric index of the icon from *icon_file* that you want to use for your self-extracting executable file. If you do not specify an index, the icon at index 0 is used.
-s	This optional parameter is available for running **ReleasePackager.exe** in silent mode. To pass this parameter to **ReleasePackager.exe**, the syntax is as follows: `ReleasePackager.exe "disk_images_folder" "package_file" ["icon_file" [icon_index]] -s` Note that the `-s` parameter must be specified after the other parameters. If you do not want to specify an icon file or icon index, you must specify the default values of a null string ("") for the icon file and 0 for the icon index.

Example

```
ReleasePackager.exe "C:\InstallShield 2016 Projects\My Project\Media\My Release\Disk Images"
    "C:\InstallShield 2016 Projects\My Project\Media\My Release\Package\MyPackage.exe"
    "C:\My Icon Files\MyIcons.dll" 2 -s
```

SetupIni.exe



Project • *This information applies to the following project types:*

- *Basic MSI*
- *InstallScript MSI*

SetupIni.exe is a command-line tool that enables you to modify a **Setup.ini** file that is embedded in a **Setup.exe** or **Update.exe** file. **Setup.ini** is an initialization file that is created during the build process of installation projects to control certain elements of the installation.

SetupIni.exe is installed in the following location:

*InstallShield Program Files Folder***System**

Syntax

SetupIni.exe *PathToSetup.exe Section Key Value*

Parameters

Table 12-11 • Command-Line Parameters for SetupIni.exe

Parameter	Description
PathToSetup.exe	Specify the path and file name to the **Setup.exe** file that contains the **Setup.ini** file that you want to modify. If the path contains spaces, enclose it within quotation marks.
Section	Specify the name of the section that you want to modify.
Key	Specify the name of the key whose value you want to set.
Value	Specify the value that you want to use for the specified key. If the value contains spaces, enclose it within quotation marks.

Example

```
SetupIni.exe "C:\InstallShield 2016 Projects\My Project\Media\My Release\Package\MySetup.exe" Startup
    CmdLine MYPROPERTY="MyValue"
```

MsiExec.exe Command-Line Parameters

MsiExec.exe is the executable program of the Windows Installer used to interpret installation packages and install products on target systems. After you build your release, you can install your Windows Installer package (.msi) from the command line.

Your Windows Installer package can be accessed from the folder that contains your built release. The default location is as follows:

C:\InstallShield 2016 Projects*ProjectNameReleaseName*\DiskImages\Disk1*ProductName*.msi**

After building a release of your product, you can install it from the command line:

```
msiexec /i "C:\InstallShield 2016 Projects\ProjectName\ReleaseName\DiskImages\Disk1\ProductName.msi"
```

The table below provides a detailed description of **MsiExec.exe** command-line parameters.

Table 12-12 • MsiExec.exe Command-Line Parameters

Parameter	Description
/i <package> or <product code>	Use this format to install the product Othello: `msiexec /i "C:\InstallShield 2016 Projects\Othello\Trial Version\Release\DiskImages\Disk1\Othello Beta.msi"` Product Code refers to the GUID that is automatically generated in the Product Code property of your product's project view.
/f [p|o|e|d|c|a|u|m|s|v] <package> or <product code>	Installing with the `/f` option will repair or reinstall missing or corrupted files. For example, to force a reinstall of all files, use the following syntax: `msiexec /fa "C:\InstallShield 2016 Projects\Othello\Trial Version\Release\DiskImages\Disk1\Othello Beta.msi"` Use the aforementioned syntax in conjunction with the following flags: • `p` reinstalls a file if it is missing • `o` reinstalls a file if it is missing or if an older version of the file is present on the user's system • `e` reinstalls a file if it is missing or if an equivalent or older version of the file is present on the user's system • `c` reinstalls a file if it is missing or if the stored checksum of the installed file does not match the new file's value • `a` forces a reinstall of all files • `u` or `m` rewrite all required user registry entries • `s` overwrites any existing shortcuts • `v` runs your application from the source and re-caches the local installation package
/a <package>	The `/a` option allows users with administrator privileges to install a product onto the network.

Table 12-12 • MsiExec.exe Command-Line Parameters (cont.)

Parameter	Description

/x <package> or <product code>	The /x option uninstalls a product.
	
	Project • *For InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler): If you try to uninstall the product through the command line using the statement* `Msiexec.exe /x {ProductCode}`*, the uninstallation may be successful. However, the Windows Installer displays an error dialog near the end of the uninstallation. The error dialog indicates that the Windows Installer service could not be accessed.*
	
	To perform an uninstallation from the command line in this scenario, the current recommended method is to use one of the following:
	
	`msiexec.exe /i {ProductCode} REMOVE=ALL`
	
	`msiexec.exe /x {ProductCode} /qn`
	
	This is not necessary if your InstallScript MSI project is using the traditional style for the InstallScript UI (which uses the InstallScript engine as an external UI handler).
	
	To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.

/j [u|m] <package> **/j [u|m] <package> /t <transform list>** **/j [u|m] <package> /g** **/j <language ID>**	Building with the /j <package> option advertises the components of your application on the end user's computer - u advertises components only to the current user - m advertises components to all users of the computer - g specifies language ID - t applies a transform to your advertised product Transforms allow the synchronization of applications across different languages. For example, if you upgrade the English version of your product, you could apply a transform to automatically upgrade the French version of your product.

Table 12-12 • MsiExec.exe Command-Line Parameters (cont.)

Parameter	Description
/L[i|w|e|a|r|u|c|m|o|p|v|x|+|!|*] <log file>	Use the `/L` option to specify the path to the log file. Optionally include one or more of the following flags to indicate which information to record in the log file: • `i` logs status messages. • `w` logs non-fatal warning messages. • `e` logs any error messages. • `a` logs the commencement of action sequences. • `r` logs action-specific records. • `u` logs user requests. • `c` logs initial user interface parameters. • `m` logs out-of-memory messages. • `o` logs out-of-disk-space messages. • `p` logs all of the properties and their values at the end of the installation. • `v` logs verbose output. • `x` logs extra debugging information. • `+` appends the information to an existing log file. • `!` flushes each line to the log. • `*` is a wild-card character that enables you to log all information (excluding the verbose output that v provides and the extra debugging information that x provides). For example, to generate a log file that logs everything verbosely, use the following: `/L*v "C:\MyLogFiles\package.log"`
/p <patch package>	Use the `/p` option to apply a patch to an installed product. To patch an installed administrative image, combine this option with `/a` as shown: `/p PatchPackage /a Package`

Table 12-12 • MsiExec.exe Command-Line Parameters (cont.)

Parameter	Description
/q [n|b|r|f]	The /q option is used to set the user interface level in conjunction with the following flags: • q or qn creates no user interface • qb creates a basic user interface The following user interface settings display a modal dialog box at the end of installation: • qr displays a reduced user interface • qf displays a full user interface • qn+ displays no user interface • qb+ displays a basic user interface
/? or /h	Either command displays Windows Installer copyright information
/y <file name>	This command calls the DllRegisterServer entry-point function of the DLL or OCX file specified in <file name>.
/z <file name>	This command calls the DllUnregisterServer entry-point function of the DLL or OCX file specified in <file name>.
/n {product code}	The /n <product code> option is available for patches that are created in the Patch Design view for Basic MSI projects that support the installation of multiple instances of a product. Use the /n option with the /p option to specify the product code of the instance to which you want to apply a patch. For example: `msiexec /p mypatch.msp /n {00000001-0002-0000-0000-624474736554}`
TRANSFORMS	Use the TRANSFORMS command-line parameter to specify any transforms that you would like applied to your base package. Your transform command-line call might look something like this: `msiexec /i "C:\InstallShield 2016 Projects\Othello\Trial Version\Release\DiskImages\Disk1\Othello.msi" TRANSFORMS="New Transform 1.mst"` You can separate multiple transforms with a semicolon. Because of this, it is recommended that you do not use semicolons in the name of your transform because the Windows Installer service will interpret those incorrectly.

Table 12-12 • MsiExec.exe Command-Line Parameters (cont.)

Parameter	Description
Properties	All public properties can be set or modified from the command line. Public properties are distinguished from private properties by the fact that they are in all capital letters. For example, COMPANYNAME is a public property. To set a property from the command line, use the following syntax: *PROPERTY=VALUE*. If you wanted to change the value of COMPANYNAME, you would enter: `msiexec /i "C:\InstallShield 2016 Projects\Othello\Trial Version\Release\DiskImages\Disk1\Othello.msi" COMPANYNAME="My Software Company"`

For information on passing **MsiExec.exe** command-line parameters through **Setup.exe**, see Setup.exe and Update.exe Command-Line Parameters.

Setup.exe and Update.exe Command-Line Parameters

Setup.exe can accept a number of command-line parameters. **Update.exe** (available only for Basic MSI and InstallScript MSI projects) accepts nearly all of the same command-line parameters. Using these parameters, end users can specify such data as the language that the installation should run in and whether to launch **Setup.exe** silently. For Basic MSI and InstallScript MSI, end users can also pass parameters through **Setup.exe** to the included .msi file.

Note • *Command-line options that require a parameter must be specified with no space between the option and its parameter. For example,* ***Setup.exe /v"ALLUSERS=2"*** *is valid, while* ***Setup.exe /v "ALLUSERS=2"*** *is not. Quotation marks around an option's parameter are required only if the parameter contains spaces. If a path within a parameter contains spaces, you may need to use quotation marks within quotation marks, as in the following example:*

Setup.exe /v"INSTALLDIR=\"c:\My Files\""

Project • *Some of the command-line options apply to only certain project types. Project-specific information is listed for each option.*

Built-In Command-Line Parameters

This section describes valid command-line parameters for **Setup.exe**. The parameters are organized into the following categories:

- Silent Installations
- Special Installation Modes
- Passing Data to the Installation
- Download and Cache Locations (Basic MSI and InstallScript MSI Projects)

- Debugging
- SMS Data
- Miscellaneous

Silent Installations

Table 12-13 • Parameters for Silent Installations

Parameter	Project Type	Description
/p"password"	Basic MSI, InstallScript MSI	If you configured the password-related settings for your release on the Setup.exe tab in the Releases view, the end user must specify the password with the /p option when running the installation silently. A typical command is as follows: `Setup.exe /s /p"password"`
/r	InstallScript, InstallScript MSI	Use this command line to run the installation in record mode. In order to run an InstallScript MSI or InstallScript installation in silent mode, you must first run **Setup.exe** with the `/r` option to generate a response file, which stores information about the data entered and options selected by the user at run time. Launching an InstallScript MSI or InstallScript installation with the `/r` parameter displays all the run-time dialogs and stores the data in a file called Setup.iss, created inside the system's Windows folder. To specify an alternative response file name and location, use the `/f1` option, described below. Basic MSI projects do not create or use a response file for silent installations.

Table 12-13 • Parameters for Silent Installations (cont.)

Parameter	Project Type	Description

/s	Basic MSI, InstallScript, InstallScript MSI	For an InstallScript MSI or InstallScript project, the command `Setup.exe /s` runs the installation in silent mode, by default based on the responses contained in a response file called **Setup.iss** in the same directory. (Response files are created by running **Setup.exe** with the `/r` option.) To specify an alternative file name or location of the response file, use the /f1 option.
		
		The command `Setup.exe /s` also suppresses the **Setup.exe** initialization dialog for a Basic MSI installation, but it does not read a response file. To run a Basic MSI installation silently, use the following command line:
		
		`Setup.exe /s /v/qn`
		
		To specify the values of public properties for a silent Basic MSI installation, you can use a command such as the following one:
		
		`Setup.exe /s /v"/qn INSTALLDIR=D:\Destination"`
		
		Note • *Using this command-line parameter to launch an installation that includes an InstallShield prerequisite does not automatically run the prerequisite installation silently. You may also need to specify a valid silent command-line parameter for the InstallShield prerequisite in the* ***Specify the command line for the application when the setup is running in silent mode*** *setting on the Application to Run tab in the InstallShield Prerequisite Editor.*
		
		For more information, see Specifying Command-Line Parameters for an InstallShield Prerequisite.

/f1	InstallScript, InstallScript MSI	Using the `/f1` option enables you to specify where the response file is (or where it should be created) and what its name is, as in the following example: `Setup.exe /s /f1"C:\Temp\Setup.iss"` Specify an absolute path; using a relative path gives unpredictable results. The `/f1` option is available both when creating a response file (with the `/r` option) and when using a response file (with the `/s` option).

Table 12-13 • Parameters for Silent Installations (cont.)

Parameter	Project Type	Description
/f2	InstallScript, InstallScript MSI	When running an InstallScript MSI or InstallScript installation in silent mode (using the `/s` option), the log file Setup.log is by default created in the same directory and with the same name (except for the extension) as the response file. The `/f2` option enables you to specify an alternative log file location and file name, as in the following example: `Setup.exe /s /f2"C:\Setup.log"` Specify an absolute path; using a relative path gives unpredictable results.

Special Installation Modes

Table 12-14 • Parameters for Special Installation Modes

Parameter	Project Type	Description
/a	Basic MSI, InstallScript MSI	 ***Note*** • *The /a parameter does not work with* **Update.exe**. **Update.exe** *launches a patch that accesses and modifies an existing cached .msi file on the system, and an administrative installation does not cache the .msi file.* The /a option causes **Setup.exe** to perform an administrative installation. An administrative installation copies (and uncompresses) your data files to a directory that is specified by the user, but it does not create shortcuts, register COM servers, or create an uninstallation log.  ***Tip*** • *If an installation contains InstallShield prerequisites and you want to extract them from* **Setup.exe**, *add a path after the* /a *parameter to extract the prerequisites to that location. A sample command is* `Setup.exe /a"C:\temp"`.
/j	Basic MSI, InstallScript MSI	The `/j` option causes **Setup.exe** to perform an advertised installation. An advertised installation creates shortcuts, registers COM servers, and registers file types, but does not install your product's files until the user invokes one of these "entry points."
/x	Basic MSI, InstallScript MSI	The `/x` option causes **Setup.exe** to uninstall a previously installed product.

Table 12-14 • Parameters for Special Installation Modes (cont.)

Parameter	Project Type	Description
/uninst	InstallScript, InstallScript MSI (if the InstallScript user interface (UI) style is the traditional style, which uses the InstallScript engine as an external UI handler)	***Important*** • *This parameter does not apply to InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.* The `/uninst` option is provided for compatibility with earlier versions of InstallShield. This option causes the InstallScript engine to execute the event handler function OnUninstall, whose default code uninstalls the previously installed product. ***Project*** • *Note that in InstallScript MSI installations, the* `/uninst` *parameter does not invoke the InstallScript engine uninstall behavior that processes install log and removes any resources that were created and logged through script. However, the* `/removeonly` *option does invoke this behavior.*
/removeonly	InstallScript, InstallScript MSI	The `/removeonly` option sets the REMOVEONLY system variable equal to a non-zero value. The default code for the OnMaintUIBefore event handler function conditionally displays the **SdWelcomeMaint** dialog, depending on the value of REMOVEONLY.

/instance=<InstanceId>	Basic MSI	The `/instance=<InstanceId>` option—which is available for Basic MSI projects that support the installation of multiple instances of a product—lets you specify which instance you want to install, update, or uninstall. <*InstanceId*> represents the value of the **InstanceId** property that identifies the instance. Whenever you use this option and include a valid **InstanceId** value, the installation suppresses the instance selection dialog.
		
		For example, the following sample command line installs the instance that has 2 as the value of the **InstanceId** property:
		
		`Setup.exe /instance=2`
		
		Use *default* to identify the instance that is installed by the base installation package, as in the following example:
		
		`Setup.exe /instance=default`
		
		To specify the instance that you want to uninstall, include the /x option with the `/instance=<InstanceId>` option.
		
		For more information, see Naming an Instance and Run-Time Behavior for Installing Multiple Instances of a Product.

Passing Data to the Installation

Table 12-15 • Parameters for Passing Data to the Installation

Parameter	Project Type	Description

/v	Basic MSI, InstallScript MSI	Use the `/v` option to pass command-line options and values of public properties through to **Msiexec.exe**.
		
		If you want to pass more than one argument to **Msiexec.exe**, you can use the `/v` option multiple times at the command line, once for each argument. For example:
		
		`Setup.exe /v"/l*v c:\test.log" /v"MYPROPERTY1=value1" /v"/qb"`
		
		As an alternative, you can pass multiple arguments through the `/v` option as in the following example:
		
		`Setup.exe /v"/l*v c:\test.log MYPROPERTY1=value1 /qb"`
		
		
		
		Note • *If you pass the `/v` parameter at the command prompt when launching* **Setup.exe**, *any parameters that are specified for the CmdLine keyname in* **Setup.ini** *are ignored. To learn more, see Setup.ini.*

/ v"ISSCRIPTCMDLINE=\"<Option1> <Option2>\""	Basic MSI projects that have InstallScript custom actions	This option specifies command-line parameters to be passed to the script. Any property supported by InstallScript MSI (where appropriate) can be specified. (The most common ones are `/d` and `/z`.) For example, the following indicates that you want to debug the script, and that the *CMDLINE* variable should contain **TEST**. `Setup.exe /v"ISSCRIPTCMDLINE=\"-d -zTEST\""` Note that as shown above, when you want to specify that a double quote character is not a delimiter for the command line but a delimiter for the property, use **\"**. Note also that as with any public Windows Installer property, this property should be specified with all uppercase letters.
/z	InstallScript MSI	The `/z` option is used to pass data to the InstallScript system variable CMDLINE, as in the following example: `Setup.exe /z"My Custom Data"` When this command line is used, the variable *CMDLINE* contains the string **My Custom Data**.

Download and Cache Locations (Basic MSI and InstallScript MSI Projects)

Table 12-16 • Parameters for Download and Cache Locations

Parameter	Project Type	Description
/ua **/uw**	Basic MSI, InstallScript MSI	In the Release Wizard, you can specify download locations for the Windows Installer installers **InstMsiA.exe** and **InstMsiW.exe**. A user can specify an alternative URL at run time using the `/ua` option (for the **InstMsiA.exe** location) and `/uw` option (for the **InstMsiW.exe** location), as in the following example: `Setup.exe /uw"http://www.otherlocation.com/engines"` The file name is not necessary.  ***Note*** • *You must specify the full URL with the parameters.*
/um	Basic MSI, InstallScript MSI	In the Release Wizard, for a Web Downloader build, you can specify a download location for your .msi database. A user can specify an alternative URL using the `/um` option, as in the following example: `Setup.exe /um"http://www.otherlocation.com/packages/product.msi"`  ***Note*** • *You must specify the full URL with the parameter.*
/b	Basic MSI, InstallScript MSI	In the Release Wizard, for a Downloader build, you can specify whether to cache the contents of a compressed package on the local system. With the `/b` option, the user can specify the directory in which to cache the installation files, as in the following example: `Setup.exe /b"C:\CacheDirectory"`  ***Note*** • *You must specify the full URL with the parameter.*

Debugging

Table 12-17 • Parameters for Debugging

Parameter	Project Type	Description
/d	Basic MSI projects with InstallScript custom actions, InstallScript, InstallScript MSI	For an InstallScript project, running the command `Setup.exe /d` runs the installation program with the InstallScript Debugger. ***Note*** *• Debugging InstallScript code requires the debug-information file* **Setup.dbg** *to be available. In addition, in order to debug an installation on a system other than your development machine, you need to copy certain files from your development machine to the debugging machine. To learn more, see Debugging an Installation on Any Computer.* For a Basic MSI project, the following command runs your InstallScript custom actions in the InstallScript Debugger: `Setup.exe /v"ISSCRIPTDEBUG=1 ISSCRIPTDEBUGPATH=\"path-to-Setup.dbg\""`

/debuglog	Basic MSI, InstallScript MSI	The `/debuglog` parameter lets you generate a log file for **Setup.exe**.
		
		To generate a log named **InstallShield.log** in the same directory as the **Setup.exe** file, pass just the command-line parameter. Note that this does not work if the **Setup.exe** file is in a read-only location. For example:
		
		`setup.exe /debuglog`
		
		To specify the name and location of the log file, pass the path and name, as in the following example:
		
		`Setup.exe /debuglog"C:\PathToLog\setupexe.log"`
		
		To generate a log file for the feature prerequisites in the installation, use the `/v` parameter to set the **ISDEBUGLOG** property to the full path and file name for the log file, as follows:
		
		`Setup.exe /debuglog"C:\PathToSetupLogFile\setup.log" /v"ISDEBUGLOG=prereq.log"`
		
		You can use directory properties and environment variables in the path for the feature prerequisite log file.

SMS Data

Table 12-18 • Parameters for SMS Data

Parameter	Project Type	Description
/m	InstallScript, InstallScript MSI	The `/m` option causes **Setup.exe** to generate an SMS Management Information Format (.mif) file. Following is a typical command: `Setup.exe /m"SampleApp"` Including the ".mif" file extension is not necessary.
/m1	InstallScript, InstallScript MSI	Using the `/m1` parameter (along with `/m`) enables you to specify a serial number to be written to the .mif file. A typical command is as follows: `Setup.exe /m"SampleApp" /m1"1234-5678"`
/m2	InstallScript, InstallScript MSI	Using the `/m2` parameter (along with `/m`) enables you to specify a locale string to be written to the .mif file. A typical command is as follows: `Setup.exe /m"SampleApp" /m2"ENU"`

Miscellaneous

Table 12-19 • Parameters for Miscellaneous

Parameter	Project Type	Description
/delayedstart:<number of seconds>	InstallScript	Specifies the amount of time (in seconds) by which initialization of the installation is delayed after **Setup.exe** is launched.  ***Note*** • *Using the -delayedstart option is recommended when manually launching an additional installation after reboot (for example, by using the RunOnce key). The delay allows the operation system to initialize completely; this prevents the problems—such as Remote Procedure Call (RPC) errors—that can occur if an installation initializes before the operating system has initialized completely. The recommended delay length is 30 seconds.* *Note that this option is not needed when the installation starts automatically after reboot (for example, due to a call to* ***SdFinishReboot*** *before reboot).*

Table 12-19 • Parameters for Miscellaneous (cont.)

Parameter	Project Type	Description
/runfromtemp	Basic MSI, InstallScript, InstallScript MSI	This parameter allows the setup author to always clone the setup and run it from the temporary directory, even if the setup does not meet the conditions for running from the temporary directory. This parameter is ignored if the setup is a self-extracting executable file (.exe).
/clone_wait	InstallScript	This parameter indicates that the original setup should wait for the cloned setup process to complete before exiting.
/extract_all:<path>	InstallScript	Specifies that a self-extracting package's files should not be run but simply extracted to the location that is specified by <path>.
/h	Basic MSI, InstallScript MSI	The build engine automatically creates an installation that supports **Setup.exe** cloning in cases where cloning is required (for example, multi-disk installations). If you need to do this manually, pass `/h` to **Setup.exe** and it will clone itself to a temporary location and run from that location.

Table 12-19 • Parameters for Miscellaneous (cont.)

Parameter	Project Type	Description
/hide_progress	Basic MSI, InstallScript, InstallScript MSI	Suppresses the display of any small and standard progress dialogs that might be shown during initialization. The small progress dialog is usually used for installations that display a splash screen during initialization, since a standard-size progress dialog does not leave any space for the splash screen. Specifying the `/hide_progress` option hides the small progress dialog for those installations, so end users would see just the splash screen without any progress indication.  ***Note*** • *For InstallScript installations: If you specify* `/hide_progress` *and include a splash screen in your InstallScript installation, the length of time that the splash screen is displayed depends on whether* **SmallProgress=Y** *or* **SmallProgress=N** *is specified in* **Setup.ini**. • *If* **SmallProgress=Y** *is specified, the splash screen is shown for as long as the progress dialog would have been displayed if* `/hide_progress` *was not specified.* • *If* **SmallProgress=N** *is specified, the splash screen is shown for the length of time specified in the SplashTime key; thus, using* `/hide_progress` *and* **SmallProgress=Y** *at the same time is not recommended.* *For InstallScript MSI installations: If you include a splash screen, the installation automatically switches to the small progress dialog, and the splash screen is shown only during the time that the progress dialog is displayed. Note that this is true even if* `/hide_progress` *is specified. Therefore, it is recommended that you avoid using* `/hide_progress` *with a splash screen in InstallScript MSI installations.*
/hide_splash	InstallScript, InstallScript MSI	Suppresses the display of the splash screen if one is included.
/hide_usd	InstallScript	Suppresses display of the dialog that is displayed by an update-enabled installation to let the end user select which instance of your product will be updated. This dialog is displayed by default when an update-enabled installation detects multiple previous instances. When this command-line option is used and an update-enabled installation detects multiple previous instances, the installation creates a new instance.

Table 12-19 • Parameters for Miscellaneous (cont.)

Parameter	Project Type	Description
/ig{InstanceGUID}	InstallScript	Specifies the value of the system variable INSTANCE_GUID; for example, -ig{722C7440-B317-4B3B-AECA-0199EA4E7CDB}. If this option is not used, the installation automatically assigns a value to INSTANCE_GUID (for multi-instance installations, this value is a newly generated GUID; for standard installations, this value is the same as the value of PRODUCT_GUID). This option is useful if you have created an installation launcher—that is, a custom application that runs before your installation does to perform pre-setup tasks, such as determining the instance GUID that you want to use for the installation. Do not specify anything other than a valid GUID with this option.
/installfromweb"PathToDisk1Files"	InstallScript	This option is similar to the media_path option except that this option forces the installation to behave like a launched One-Click Install installation, even if the path to the media files is not a URL. Use this option if you are launching the installation from a Web page manually. In addition, this option is added automatically if the built-in **Setup.ocx** file is used to launch the installation.
/media_path"PathToDisk1Files"	InstallScript	This option indicates that the installation should look for the Disk1 files in the location that is specified. Note that only the **Setup.exe** file needs to be in the original launch location; the installation obtains all other required files from the specified location (including **Setup.exe**, which must be present in the media location). You can specify a URL as the path to the media files; in this case, the installation behaves like a launched One-Click Install installation, which always shows the security dialog. To learn more about the behavior of One-Click Install installations, see One-Click Install Installations in InstallScript Projects.

/L*<LanguageID>*	Basic MSI, InstallScript, InstallScript MSI	This option indicates that the installation should run in the specified language as specified. You can specify the language ID as either a hexadecimal or decimal number. If you specify the hexadecimal number, be sure to proceed the value with **0x**. For example, the following commands indicate that the installation should be run in German:
		`Setup.exe -L0x0407`
		`Setup.exe -L1031`
		Note that if you specify a language ID that is not supported by the installation or you specify an invalid language ID, the parameter is ignored. Also note that if this parameter is specified and it is valid, the language dialog (if enabled) is automatically suppressed.

Table 12-19 • Parameters for Miscellaneous (cont.)

Parameter	Project Type	Description
/w	Basic MSI, InstallScript MSI	For a Basic MSI project, the `/w` option forces **Setup.exe** to wait until the installation is complete before exiting. ***Note*** • *If you are using the `/w` option in a batch file, you may want to precede the entire Setup.exe command-line option with `start /WAIT`. A properly formatted example of this usage is as follows:* `start /WAIT setup.exe /w`
/SMS	InstallScript MSI	For an InstallScript MSI project, **Setup.exe** automatically waits for the installation to finish before exiting, so this option (used by earlier versions of InstallShield Professional) is no longer necessary.

User-Defined Command-Line Parameters

Project • *This information about user-defined command-line parameters applies to InstallScript projects.*

For user-defined command-line parameters in InstallScript MSI projects, use the `-z` command-line parameter that is described above.

Along with the command-line parameters that are listed above, `-bd`, `-f`, and `-zi` are command-line parameters that are reserved for use in InstallScript projects. User redefinition of these command-line parameters, either uppercase or lowercase, can cause errors.

You can define your own command-line arguments, which are copied to the system variable CMDLINE as at run time. Like predefined command-line parameters, you can pass custom arguments directly to **Setup.exe**, place them in **Setup.ini**, or (for testing purposes while you are using the InstallShield IDE) place them in the Settings dialog box, which is displayed when you click Settings on the Build menu in InstallShield.

Note • **Setup.exe** *initializes correctly even on systems with more than 256 MB of memory and always stays in memory until the setup is complete. Due to the nature of DOS, when you launch* **Setup.exe** *from the command line, a DOS prompt is quickly returned although* **Setup.exe** *is still in memory.*

Advanced UI and Suite/Advanced UI Setup.exe Command-Line Parameters

Project • *This information applies to the following project types:*

- *Advanced UI*

- *Suite/Advanced UI*

***Edition** • The Advanced UI project type is available in the Professional edition of InstallShield. The Suite/Advanced UI project type is available in the Premier edition of InstallShield. For information about the differences between these two project types, see Advanced UI Projects vs. Suite/Advanced UI Projects.*

The **Setup.exe** setup launcher for Advanced UI and Suite/Advanced UI installations can accept a number of command-line parameters. Note that passing undefined parameters or invalid property definitions results in an invalid command-line error.

Table 12-20 • Parameters for Advanced UI and Suite/Advanced UI Setup Launchers

Parameter	Description

/d
 	
/d"PathTo.dbgFile"
	The `/d` parameter lets you debug the InstallScript actions in your Suite/Advanced UI installation with the InstallScript Debugger.
	
	***Note** • Debugging InstallScript code requires the debug-information file* **Setup.dbg** *to be available. In addition, in order to debug an installation on a system other than your development machine, you need to copy certain files from your development machine to the debugging machine. To learn more, see Debugging an Installation on Any Computer.*
	
	The following command runs your InstallScript custom actions in the InstallScript Debugger:
	
	`Setup.exe /d`
	
	If the installation is located in a path at which they were originally compiled, you also need to specify the path to your debug symbols file (.dbg):
	
	`Setup.exe /d"Path-to-Setup.dbg"`
	
	For more information about InstallScript actions in Suite/Advanced UI projects, see Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation.
	
	For more information on debugging issues, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.

Table 12-20 • Parameters for Advanced UI and Suite/Advanced UI Setup Launchers (cont.)

Parameter	Description

/debuglog
/debuglog"PathToLog"
	The `/debuglog` parameter lets you generate a log file for **Setup.exe**.
	To generate a log named **InstallShield.log** in the same directory as the **Setup.exe** file, pass just the command-line parameter. Note that this does not work if the **Setup.exe** file is in a read-only location. For example:
	`Setup.exe /debuglog`
	To specify the name and location of the log file, pass the path and name, as in the following example:
	`Setup.exe /debuglog"C:\PathToLog\setupexe.log"`
	For more information, see Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation.

/language:LCID	The `/language` parameter indicates that the installation should run in the specified language as specified. Specify the language ID as a decimal number. For example, to indicate that the installation should be run in German, pass the language ID as follows: `Setup.exe /language:1031` Note that if you specify a language ID that is not supported by the installation or you specify an invalid language ID, the parameter is ignored. Also note that if this parameter is specified and it is valid, the language wizard page (if enabled) is automatically suppressed.

/log
/log"PathToPackageLog"
/log:"PathToPackageLog"
	The `/log` parameter lets you generate a log file for each package in the Advanced UI or Suite/Advanced UI installation for which logging is enabled.
	To generate package log files in a particular folder, pass the folder path with the `/log` parameter to the Advanced UI or Suite/Advanced UI **Setup.exe** file. You can optionally separate the `/log` parameter and the path with a colon. For example, both of the following command lines generate a log file in the **PathToLogFiles** folder:
	`Setup.exe /log"C:\PathToLogFiles"`
	`Setup.exe /log:"C:\PathToLogFiles"`
	Note that the path to the log file location must already exist.
	To generate package log files in the %TEMP% directory, leave out the path. For example:
	`Setup.exe /log`
	Note that logging must be enabled for a package, and the package's logging-related settings (which vary, depending on whether the package is an .msi package, an .msp package, some other type of package) must be configured in the Packages view. For more information, see Supporting the Creation of Package Log Files for Command-Line Launching of an Advanced UI or Suite/Advanced UI Installation.

Table 12-20 • Parameters for Advanced UI and Suite/Advanced UI Setup Launchers (cont.)

Parameter	Description
/password:*MyPassword*	If you configured the password-related settings for your release on the Setup.exe tab in the Releases view, the end user must specify the password with the /password parameter at run time during a silent installation. A typical command is as follows: `Setup.exe /silent /password:MyPassword`

PropertyName=Value
/PropertyName:Value
PropertyName=
	You can pass properties to the Advanced UI or Suite/Advanced UI installation in more than one way. Following are examples:
	`Setup.exe MyPropertyName=MyPropertyValue`
	`Setup.exe /MyPropertyName:MyPropertyValue`
	To clear the value of a property, even if it is set by the Advanced UI or Suite/Advanced UI **Setup.xml** file, use the following syntax:
	`Setup.exe MyPropertyName=`
	The following properties have special meaning to the Advanced UI or Suite/Advanced UI installation when they are passed on the command line:
	• **ISFeatureInstall**—This property provides a comma-delimited list of feature names that are to be installed with the current installation operation. It is intended for a silent first-time installation.
	• **ISFeatureRemove**—This property provides a comma-delimited list of feature names that are to be removed with the current installation operation. It is intended for maintenance installations.
	For more information, see Advanced UI and Suite/Advanced UI Property Reference.

/remove	The `/remove` parameter uninstalls the packages that are currently installed on the target system and that were installed by the Advanced UI or Suite/Advanced UI installation. This option also removes the Advanced UI or Suite/Advanced UI installation's entry in Add or Remove Programs if the maintenance condition in the Advanced UI or Suite/Advanced UI **Setup.xml** file is false.
/repair	The `/repair` parameter repairs the packages that are currently on the target system and that were installed by the Advanced UI or Suite/Advanced UI installation.
/silent	The `/silent` parameter runs the installation without a user interface.

Table 12-20 • Parameters for Advanced UI and Suite/Advanced UI Setup Launchers (cont.)

Parameter	Description
/stage_only	The `/stage_only` parameter runs the staging part of the Advanced UI or Suite/Advanced UI installation, in which the Advanced UI or Suite/Advanced UI installation copies its packages to a directory that is specified by the user on the BrowseStageFolder wizard page. If any of the packages are compressed into the Advanced UI or Suite/Advanced UI **Setup.exe** file, the `/stage_only` option also extracts the packages from the **Setup.exe** file before copying them to the staging directory. In addition, if any of the packages are located on the Web, the `/stage_only` option downloads the packages and adds them to the staging directory. Note that this option does not run the packages in the Advanced UI or Suite/Advanced UI installation. Furthermore, it does not uncompress any of the packages. Thus, if you are creating a patch for an Advanced UI or Suite/Advanced UI installation and you need to generate an uncompressed .msi package for a package in your Advanced UI or Suite/Advanced UI installation, for example, use the stage-only option. Then perform an administrative installation for the .msi package.  ***Tip*** • *The* ***ISRootStagePath*** *property stores the path to the folder that contains the copied files. To specify a default value for this property, set this property from the command line while passing the /stage_only parameter, as shown in the following example:* `Setup.exe /stage_only ISRootStagePath="C:\MyStagingArea"`

For information on how to specify command lines that you want an Advanced UI or Suite/Advanced UI installation to use when launching one of its packages, see Passing Command-Line Parameters to a Package in an Advanced UI or Suite/Advanced UI Installation.

Setup.exe (InstallScript Projects)

Setup.exe is the main installation executable file; it performs installation initialization and launches the installation engine to execute the installation on the target system.

When you create and build a release, **Setup.exe** and other setup engine redistributable files are placed in the Disk1 folder. You must ship these files on Disk1 of your distribution media.



Note • *You can rename* **Setup.exe** *to any valid file name, such as* ***Install.exe****. Note that for the installation to be launched by another installation using the* ***DoInstall*** *function, the launched installation's* **Setup.ini** *file must have the new file name in the [Startup] section's LauncherName key.*

You can pass command-line options directly to **Setup.exe**, or you can place them in **Setup.ini**. For testing purposes, while you are using InstallShield, you can pass command-line options to **Setup.exe** through the Run/Debug tab of the Settings dialog box, which is displayed when you click Settings on the Build menu.



***Note** • A self-extracting executable file accepts any command-line parameter that* **Setup.exe** *accepts.*

For a list of supported parameters, see Setup.exe and Update.exe Command-Line Parameters.

13

Frequently Asked Questions

This section lists common questions (sorted by topic) and provides links to answers. Where the techniques or steps to perform a specific task are different in different project types, answers for each project type are displayed.

Project Types

- Which project type should I use?
- How do I convert my project to a different project type?
  - Why do I get "error C8025 'INSTALLDIR': undefined identifier" when compiling my script after conversion?
  - Why do I get the following error: "String PRODUCT_NAME was not found in string table"?

Files and Folders

- What is the difference between INSTALLDIR and TARGETDIR?
- How does Windows Installer determine whether to overwrite an existing file?
- How do I register a COM server?
- How do I install, start, stop, delete, uninstall, or configure a Windows service?
- How do I create an empty folder?
- How do I set a key file for a dynamic file link?
- What happens if I put more than one executable file in the same component?

Features and Components

- How do I conditionally select a feature?
- How do I conditionally hide a feature?
- Why should I use uppercase directory identifiers when setting component destinations?
- How do I change the value in the Link To column for a component?

Redistributables

- How do I add the Windows Installer to my project?
- How do I add the .NET Framework to my project?

Shortcuts

- How do I create an Internet shortcut?
- How do I create a shortcut to a folder?

Registry

- How do I read data from the registry?
- How do I write a property's value to the registry?

InstallScript

- How do I find out what sort of built-in functions are available in the InstallScript language?
- What is the maximum number of statements that I can use in a compiled script file? What other limits exist for InstallScript?

INI Files

- How do I read data from an .ini file?
- How do I get all key names from an .ini file?

Properties

- How do I get or set a Windows Installer property in InstallScript?
- Why does my deferred custom action show a blank value for a Windows Installer property?

Conditions

- How do I detect administrator privileges?
- How do I detect a first-time installation?
- How do I call a function only during the installation sequence?
- How do I detect if the user has selected a specific feature?
- How do I detect if the user is running a particular operating system?

End-User Interface

- How do I populate a list box at run time?
- How do I display a file browse dialog?
- How do I display a network browse dialog?

Custom Actions

- How do I search for a file on the end user's system?
- How do I launch my application after the installation is complete?
- How do I place a file in the .msi database and extract it during run time?
- How do I exit the installation from within a custom action?
- How can I use the INSTALLDIR variable after I have created a custom action that runs VBScript code?
- How can I change ODBC properties (like DBQ and SystemDB) through script?

SQL Scripts

- How do I override the default SQL run-time behavior?
- How can I control the execution of a SQL script based on certain conditions in an InstallScript project?
- How do I enforce a server-side installation in an InstallScript project?
- How do I enforce a SQL server-side installation in a Windows Installer-based project?
- How do I handle SQL run-time errors?
- How do I ensure that my SQL scripts run only against a full of SQL Server?

Building Releases

- How do I build my release from the command line?
- How do I change project settings from the command line?

Deployment

- How do users run my installation in silent mode?

Uninstallation

- How do I remove files that were created by my product?
- How do I remove registry data created by my product?

Answers to many other questions are available in the Knowledge Base.

Glossary

Table -1 • InstallShield Glossary

Term	Definition
.bin file	A binary file that is executable on UNIX platforms. Application files usually have a **.bin** extension.
.bmp file	A bitmap file of an image represented as an array of bits. In bitmap graphics, an image is displayed on the screen as a collection of tiny squares called pixels, which together form a pattern. Each pixel in the image corresponds with one or more bits.
.cab file	See cabinet file.
.command file	A command file that is executable on Mac OS X platforms. Application files usually have a **.command** extension.
.cub file	A validation module that stores and provides access to custom actions for internal consistency evaluators (ICEs). See also custom action and internal consistency evaluator.
.exe file	An executable binary file on Windows platforms. Application files usually have an **.exe** file extension.
.gif file	A graphics interchange format file supported on the Web. This type of file is in a bit-mapped graphics file format used for displaying bitmap images. These files use lossless compression, a method of compression in which no data is lost. This type of file supports up to 256 colors; it also supports transparency, where the background color can be set to transparent to let the color on the underlying page show through. This type of file format is more suitable than the **.jpg** file format for images with only a few distinct colors, such as line drawings.
.idt file	An exported Windows Installer database table.

Table -1 • InstallShield Glossary

Term	Definition
.ism file	The working file that InstallShield uses to store project information. When you build a release, InstallShield uses the **.ism** file to create an **.msi** file for distribution.
.jar file	A Java archive file that contains the classes, images, and sound files for an installation, zipped up into a single file.
.jpg file	Also known as JPEG, which is the abbreviation for Joint Photographic Experts Group, the original name of the committee that wrote the standard. This type of file is one of the image file formats supported on the Web. The file uses lossy compression, a type of compression in which color and grayscale continuous-tone images are compressed and some data is lost to eliminate redundant or unnecessary information. These images support 16 million colors and are best suited for photographs and complex graphics.
.mif file	A management information format file used to describe a hardware or software component.
.msi file	The Windows Installer installation package in its finished state. It includes installation resource files and can have compressed within it all of the application's data files. The **.msi** file is the one that is distributed to the end users and will interact with the Windows Installer service to install your application on their machines.
.msm file	The database file of a merge module. This type of file contains all of the installation properties and installation logic for the module. See also merge module.
.msp file	A Windows Installer patch file. This type of file consists of a summary information stream, transform substorages, and cabinet files. An **.msp** file contains at least one database transform that adds patching information to the database of its target installation package. The installer uses this patching information to apply the patch files that are stored in the cabinet files.
.mst file	A transform package. This type of file is a simplified Windows Installer database that contains the differences between two **.msi** databases. Transforms enable administrators to apply modified settings to a database when they are deploying an installation package. See also transform.
.NET	An operating system platform that Microsoft created for connecting information, people, systems, and devices. The .NET environment enables developers to build, create, and deploy their applications and Web services using whatever languages they prefer via the common language run time.
.NET Compact Framework	A scaled-down version of the .NET Framework designed to run on resource-constrained devices, such as mobile devices. The .NET Compact Framework is currently available only for Pocket PC and Windows CE .NET devices, with support for more devices forthcoming.

Table -1 • InstallShield Glossary

Term	Definition
.NET Framework	A programming infrastructure that Microsoft created for building, deploying, and running applications and services that use .NET technologies such as Web services. The .NET Framework consists of three main parts: the common language run time, a hierarchical set of unified class libraries, and a componentized version of Active Server Pages called ASP.NET.
.ocx file	An object linking and embedding (OLE) custom control file. This type of file is a cross-platform COM file that is called by an application to perform a function, such as the ability to resize windows.
.p12 file	A certificate file that conforms to public-key cryptography standard (PKCS) number 12, which specifies a portable format for securely storing or transporting a user's private keys, certificates, miscellaneous secrets, and other information.
.pcp file	A Windows Installer patch creation properties file.
.prq file	A prerequisite file that contains information about a base application or component that must be installed on the target machine before the main application can be installed. Including setup prerequisites in installation projects enables developers to chain multiple installer files together into a single executable file.
.reg file	See registry file.
.scm file	A ScreenCam movie file.
.wmf file	A Windows metafile format file. This type of graphics file is used to exchange graphics information between Microsoft Windows-based applications.
absolute path	An absolute path includes all of the information necessary to locate a file by starting at the root directory of a specified drive. For example, **C:\Program Files\InstallShield** is the absolute path to the InstallShield folder when it is installed on drive C.
accessibility	The capacity or tendency to be available to all people, including those with disabilities.
acquisition phase	The phase of the installation process during which the installer queries the database for instructions. The acquisition phase is followed by the execution phase.
action	A command that performs an operation at a particular point during the execution of an installation or uninstallation wizard. Developers of installation packages can use built-in standard actions and create their own custom actions. Actions may display progress to the end user or allow the end user to cancel the operation.
active directory	A structure supported by Microsoft Windows 2000 that enables administrators to track and locate any object on a network. Active Directory is the directory service used in Windows 2000 Server for distributed computing environments.

Table -1 • InstallShield Glossary

Term	Definition
active template library (ATL)	A set of classes for writing COM controls, resulting in smaller binaries than, for example, MFC.
ActiveX	A set of object-oriented programming technologies and tools. ActiveX enables developers to write applications so that other applications and the operating system can call them. ActiveX technology makes it possible to create interactive Web pages that look and behave like applications, rather than static pages.
administration	The act or process of managing something.
administration sequence	The list of actions that are executed when a user launches your installation with the `/a` command-line option. This is useful for network administrators who want to provide a common installation point on the network and prepare different installation criteria for multiple users. Each Administration sequence consists of a User Interface sequence and an Execute sequence.
administrative installation	Copies and uncompresses your data files to a directory specified by the user, but does not create shortcuts, register COM servers, or create an uninstallation log.
administrative privileges	The highest level of permission that can be granted to a computer user. Levels of permissions are necessary in networked environments to ensure system security and prevent damage to computer hardware and software. A user with administrative privileges can perform tasks such as install and uninstall software and change a computer's configurations. Administrative privileges usually pertain to Windows NT 4.0, 2000, or XP machines as opposed to Windows 95, 98, or ME machines.
advertised shortcut	A shortcut to a product or feature that is not installed until the first time that the end user launches the shortcut. At run time of an installation, if the end user selects the "This feature will be installed when required" option for the product or the feature containing the shortcut, the shortcut is created but the component's files are not installed until the end user launches the shortcut. The first time the shortcut is launched, the Windows Installer service installs the component's files and other data and then the shortcut launches the target file. Every time the shortcut is used from then on, it behaves like a normal shortcut.
advertisement	A type of "just-in-time" installation in which features are not installed immediately during the installation process. Instead, they are installed on the fly when they are requested. When you enable feature advertisement, the feature is advertised, regardless of the mode in which the installation is running, as long as no other factors prevent it from being advertised. If you launch **MsiExec.exe** from the command line with the `/jm` option, the feature is advertised to the end user's machine. If you use the `/ju` function, the feature is advertised to the current end user. In the Custom Setup dialog, the end user can control which features are immediately installed and which are available later. The two types of advertising are assigning and publishing. See also assigning; publishing.

Table -1 • InstallShield Glossary

Term	Definition
advertisement sequence	The advertisement sequence contains the list of actions that are executed when a user launches your installation with the `/j` command-line option. Validation rule ICE78 requires the advertisement user interface sequence to be empty.
advertising	The process of presenting to end users features that are not installed immediately during the installation process. Instead, these features are installed on the fly when they are requested. See also advertisement.
AlwaysInstallElevated	A user policy that can be configured for Windows platforms under the following registry keys: HKEY_LOCAL_MACHINE\Software\Polices\Microsoft\Windows\Installer and HKEY_CURRENT_USER\Software\Polices\Microsoft\Windows\Installer. To install a package with elevated (system) privileges, set the `AlwaysInstallElevated` value under both of these registry keys to **1**.
American Standard Code for Information Interchange (ASCII)	The code that represents letters, numbers, punctuation marks, and other characters such as numbers, with each character assigned a value between 0 to 127. This code enables different types of computers and computer applications to exchange data.
animation	Multiple graphic images that are alternately changed and displayed at a specified number of frames per second to produce the illusion of movement.
API	See application programming interface.
app paths	The registry key that Windows uses to find your application and its **.dll** files if their locations are not already in the system's path. App paths entries can be set through the component's advanced settings.
application programming interface (API)	A set of routines that an application uses to communicate with the computer's operating system and that the operating system uses to make services available to the application.
ASCII	See American Standard Code for Information Interchange.
assigning	A type of advertisement. If an administrator assigns an application to a machine, the installation program automatically runs the next time that the machine starts or restarts. If the administrator assigns an application to a user, the installation program places a shortcut on the Start menu of the user's machine. When the user selects the shortcut or launches a document associated with the assigned application, the application is installed.
asynchronous execution	A custom action that independently continues with the execution of its thread while the installer executes the main installation.
ATL	See active template library.

Table -1 • InstallShield Glossary

Term	Definition
autonomic computing	A self-managed computing model named after, and patterned on, the human body's autonomic nervous system. An autonomic computing system would control the functioning of computer applications and systems without input from users, similar to the way that the human body's autonomic nervous system regulates and protects the body.
auto-repair	The automatic restoring of an application to its original state by the installer. A partial or complete application reinstallation might be required if any files associated with any feature are missing or corrupted.
basic UI	One level of the installer's internal user interface (UI) capabilities. Typically, installation packages that are built with the basic UI level display built-in modeless dialog that show progress messages and disc prompt messages. They do not display any authored dialogs.
billboards	Images, such as marketing messages, that can be displayed during installation.
Binary table	A table that holds the binary data for items such as bitmaps and icons. It also contains data for custom actions.
bitmap	A representation of an image presented as an array of bits. In bitmap graphics, an image is displayed on the screen as a collection of tiny squares called pixels, which together form a pattern. Each pixel in the image corresponds with one or more bits.
cabinet file	A single file that holds a number of compressed files. During installation of an application, the compressed files are decompressed and copied to your computer. Cabinet files are efficient because they save space and time when distributing software. A cabinet file usually has the file extension **.cab**. Missing or corrupt files may prevent installations from completing. It may be necessary to replace missing or corrupt operating system files or InstallShield files by extracting them from a cabinet file.
cache	A cache (pronounced *cash*) is a temporary storage area for frequently accessed data. The purpose of caching is to store frequently used information in a location that is easy to access, resulting in a faster-running computer. There are two types of cache: memory cache and disc cache. Memory cache stores the data and the address of where the data is stored in main memory. Memory caching is useful because most applications access the same data repeatedly. Disc caching uses the main memory. It is used to hold information that has recently been requested from the hard disc or has previously been written to the hard disc. In general, installations usually use the disc cache. When data is read from or written to main memory, a copy is also saved in the memory cache. When data is called for, the computer first checks the memory cache, then the disc cache, and finally main memory.
caption	A text heading in a window. The windows caption is what users click and drag to position the active window on the screen.

Table -1 • InstallShield Glossary

Term	Definition
CBT	See computer-based training.
CD	See compact disc.
CD browser	A graphical user interface that is launched when a CD is inserted in the drive. It is used to launch one or more applications on the CD.
CD-ROM	See compact disc read–only memory.
checksum	A calculated figure that is applied to data to test for possible corruption. The checksum is derived by sequentially combining bytes of data in the file in a systematic manner. After transmission or storage compression, the checksum calculation is performed again and the result is compared with the previous outcome. If the numbers do not match, this indicates that there is likely an error in the stored or transmitted data.
column	A vertical set of information.
COM	See component object model.
COM file	Binary software components containing reusable code that can be shared across products. A popular example of a COM file is a Microsoft Excel spreadsheet when it is embedded in a Word document. In this example, Excel acts as the COM server and Word acts as the client.
COM server	An executable file that exposes objects to other applications according to the Component Object Model (COM) specification. The types of objects exposed by a COM server may include ActiveX controls, ActiveX documents, Automation objects, or MTS components.
COM+ file	An extended COM file. With regard to installations, COM+ adds greater support for building distributed components. For example, you can create appID keys and values in the registry and thereby configure various COM components to run remotely or with special privileges.
command	(1) An instruction given to a computer. (2) An instruction that the operating system executes from the command line or a Command Prompt window to perform a specific task. If errors occur during installation, you may, for example, need to execute commands to run utilities, search your computer's directories, or delete files.
command line	The area of the Command Prompt window in which commands are typed. A command line is used to pass commands to executable files (for example,**.exe**, **.bin**).

Table -1 • InstallShield Glossary

Term	Definition
command prompt window	The Command Prompt window is the interface to MS-DOS. To access the Command Prompt window, click Run on the Start menu, and then type **command.com** in Windows 95, 98, and ME or **cmd.exe** in Windows NT, 2000, or XP. If errors occur during installation, you may need to open the Command Prompt window to enter commands to run utilities or search your computer's directories, for example.
command-line option	A command-line option is an argument to a command that changes how the command is executed. See also command.
commit execution	Execution of a Windows Installer action upon completion of the InstallFinalize action, which occurs when the installation has completed transferring files, registering COM servers, and creating shortcuts and registry entries.
committing databases	Accumulated changes made in a Windows Installer database. The changes are not reflected in the actual database until the database is committed, that is, until **MsiDatabaseCommit** is called.
compact disc (CD)	A disc used for electronically recording, storing, and playing back audio, video, and other information in digital form.
compact disc read-only memory (CD-ROM)	A type of compact disc that is only readable.
component	Elements of the application from the installation developer's perspective. Components are not visible to the end user. When the end user selects a feature for installation, the installer determines which components are associated with that feature, and then those components are installed. Components of an application would contain, for example, the executable binary files, data files, shortcuts, help system files, and registry entries.
component object model (COM)	A Microsoft-developed software architecture that enables the creation of component-based applications. Component-based applications allow other components and other applications to use their features, adding functionality to these programs.
compress	Take one or more files and create a new file whose size is less than the total size of the original file(s) and from which the original file(s) can be re-created.
computer-based training (CBT)	Any instruction delivered by computer. CBTs often use graphics, sound, animation, and interaction to train people.
condition	A statement of logic that compares the value of a property to a fixed value or determines whether the property is defined. The condition must evaluate to true if the object, feature, component, action, or other item associated with the condition is to be installed or performed.

Table -1 • InstallShield Glossary

Term	Definition
conditional installation	Installation of certain items only if certain conditions are met. For example, if an operating system condition is associated with the installation of an application, the application is installed on a target machine only if that target machine is running the specified operating system.
consume	What an installation does to a redistributable package such as a merge module or InstallShield object.
context menu	A context menu, also known as a right-click menu or a pop-up menu, opens when a user right-clicks an item on the desktop, in Windows Explorer, or in an application.
converted project	(1) A file that has been transformed from one type of file to another. (2) The Open MSI/MSM Wizard in InstallShield is a tool that converts **.msi** files and merge module (**.msm**) files to InstallShield installation projects (**.ism** files) that you can modify and build in InstallShield.
costing	File costing is the process that InstallShield uses to determine the total disc space that a current installation requires.
custom action	An action that is created by an installation author as opposed to a standard action that is built into a Windows installer. The action encapsulates a function performed during the installation, uninstallation, or maintenance of an application.
database	A discrete collection of data in a database management system (DBMS).
database function	A function that operates on a database.
database handle	A quantity that specifies a database when a database function is called.
database management system (DBMS)	A set of applications that control the organization, storage, and retrieval of data for many users.
DBMS	See database management system.
DCOM	See distributed component object model.
DCOM file	Software components that are able to communicate across a network. DCOM, previously called network OLE, allows the components for a single application to be distributed across multiple networked computers.
decompress	The reverse process of compression, which minimizes the size of a file by removing its space characters. This process returns the file to its original state.
deferred execution	Execution of an installer action upon execution of the installation script.

Table -1 • InstallShield Glossary

Term	Definition
dependency	A software object that is required by another software object.
dialog	Windows of information that display to the end user during installation and uninstallation. They enable the end user to interact with the operation by reading or specifying information. See also wizard.
dialog box	A window that displays within an application interface that enables users to enter information or specify commands.
differential release	A release that contains only those files that were absent from one or more of a specified set of existing releases. A differential release is used to update the versions of your product that were installed by those existing releases.
digital signing	To assure end users that the code within your application has not been tampered with or altered since publication, you can digitally sign your application. When you do so, end users are presented with a digital certificate when they download your product.
distributed component object model (DCOM)	An extension of the component object model (COM), a Microsoft-developed software architecture that enables the creation of component-based applications.
distribution media	A CD-ROM or other format of media deployed to users.
DLL	See dynamic link library.
drag	The act of selecting an item and moving it to another location.
dynamic link library (DLL)	A shared, code-base file containing functions that can be called from other applications.
edit field object	An interactive object that lets the user enter text into a field.
elevated privileges	Privileges that are higher than standard privileges, and are usually temporarily. Privileges are permissions to perform certain actions on a system.
end user	The person who installs or uninstalls your product.
end user machine	The machine onto which an end user installs or from which uninstalls your product.
engine	A program that performs essential functions and coordinates the overall operation of other programs. Engines work behind the scenes. InstallShield uses the InstallScript engine (**ISScript.msi**) and the Windows Installer engine to drive installations. InstallShield also has its own proprietary engine called ikernel.
environment variable	A variable that can be accessed by multiple programs on the target system.

Table -1 • InstallShield Glossary

Term	Definition
EXE media type	All of the installation and uninstallation files that are contained in a single, self-extracting file.
executable (.bin) file	A file that contains a program that can run on UNIX platforms.
executable (.exe) file	A file that contains a program that can run on Windows platforms.
execution phase	The phase during which an installer's actions are executed.
execution script	Installer actions for a Windows Installer installation. The execution script is generated during the acquisition phase of installation and executed during the execution phase.
expand	(1) To restore a compressed file to its original size. A compressed file is a single file that contains one or more other files, for example, a cabinet file. Compressed files are reduced in size, thus, saving space. (2) A DOS command used to restore compressed files. During an installation, missing or corrupted operating system files or InstallShield files may need to be expanded from a compressed file to enable the installation to complete.
extensible markup language (XML)	(1) A programming language that is essentially a simplified version of standard generalized markup language (SGML). It enables developers to create customized tags to organize and deliver content efficiently. (2) The format of the output of MultiPlatform projects. (3) Windows-based projects created in InstallShield can be converted to XML so that they can be saved in source code control software.
extensible stylesheet language (XSL)	A language used to describe how XML information should be presented.
external user interface	The user interface developed by the author of an installation package. It does not use the internal user interface capabilities that are available with the installer.
extract	To remove, or decompress, a file from a compressed file, such as a cabinet file. A compressed file is a single file that contains one or more other files. Compressed files are reduced in size, thus, saving space. To use a file that has been compressed, it must first be "pulled out" of the compressed file. A command-line utility called **Extract.exe** can be used to extract files. During installation, missing or corrupted operating system files or InstallShield files may need to be extracted from a compressed file to enable the installation to complete.
false positive	A false positive is something that gives the appearance of being true by a test, but in reality is not. Most cases of false positives are found in anti-virus software. For example, anti-virus software may claim that it has found a virus in an InstallShield file, but in fact, this is not the case; their virus definitions need to be updated.

Table -1 • InstallShield Glossary

Term	Definition
feature	Logical representations of the functionality of the product. For example, an installation could consist of a database, the main application files, and the help system files. Database, Application, and Help System would all be features of the product. Although features are visible to the end user, the actual files that comprise the features are within the components of the features. See also component.
file	An element of data storage in a file system.
file extension	The portion of a file name, following the final point, that indicates the kind of data stored in the file.
InstallShield scripting run time	See IDriver.exe.
formatted	When data has been divided or arranged for storage or display.
full update	An installation that installs an updated version of your product over an earlier version.
full user interface (UI)	One level of the installer's internal user interface (UI) capabilities. Installation packages that are built with the full UI level can display both the modal and modeless dialogs that have been included in the internal UI.
gallery	A collection of available resources that can be shared.
globally unique identifier (GUID)	A long string of numbers created by InstallShield to uniquely identify your product from others. Enter string GUIDs throughout InstallShield in the following format: {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}.
GUI	Graphical user interface.
GUID	See globally unique identifier.
hard drive	The primary storage device on a computer. A hard drive contains disks on which data is read from and written to magnetically. The term "hard" differentiates the aluminum or glass disks used in a hard drive from floppy disks, which are made of plastic.
hotfix	A quick, important fix for a bug whose deployment cannot wait until the next release of the application. Most hotfixes are in the form of small patches and can be downloaded from the software vendor's Web site.
HTML	See hypertext markup language.
hypertext markup language (HTML)	A language used to create Web pages with hyperlinks and markup for text formatting.
IAT	See import address table.

Table -1 • InstallShield Glossary

Term	Definition
ICE	See Internal Consistency Evaluator.
IDE	See interactive development environment and integrated development environment.
IDriver.exe	The InstallShield scripting run-time engine. It is required on a computer to run some installations created with InstallShield. **IDriver.exe** is located in one or both of the following common file locations by default: **C:\Program Files\Common Files\InstallShield\Driver\7\Intel 32** or **C:\Program Files\Common Files\InstallShield\Driver\8\Intel 32**. Some installation error messages may reference **IDriver.exe**.
IE	See Internet Explorer.
ikernel.exe	The InstallShield engine. See also engine.
immediate execution	Installation phase terminology that generally refers to custom actions that are immediately executed when the installer builds the script in an InstallScript project.
import address table (IAT)	The table used to import **.dll** files or executable files.
indirect build	A full build that creates references to the payload files only. This means that at installation time, the payload must be in the same place where it was at build time. For example, if your payload is on a network server, then the installation looks for the payload on that same network server.
install level	A feature's install level partly determines whether it is selected by default for installation. That value is compared to the Windows Installer property INSTALLLEVEL to determine which features are selected for installation: if a feature's install level property is less than or equal to the value of the INSTALLLEVEL, the feature will be installed.
installation	The transfer of a program and its constituent files, features, and components from source media to a target system. See also installation process.
installation package	One or more files that are used together to install or uninstall an application.
installation process	The entire process of installing the contents of a package or application onto a target computer. The installation process consists of the acquisition phase, the execution phase, and, if the installation attempt is not successful, rollback.
installation project	The entire collection of source files, dialogs, actions, and conditions that make up your installation and uninstallation while it is under construction.
installation sequence	The sequence in which an applications files are installed, acitons are executed, and dialogs display at run time. The installation sequence runs by default when the installation is launched, for example, by double-clicking the installation launcher.

Table -1 • InstallShield Glossary

Term	Definition
installation type	A predefined group of features from which the end user can select to install or uninstall.
installation-on-demand	A Windows Installer capability that makes it possible to offer functionality to users and applications in the absence of the files themselves.
INSTALLDIR	A system variable that specifies the root destination directory for an installation.
installer	An installation and configuration service on a computer.
installer database	A database that contains all the necessary information for the installation of an application. It consists of many interrelated tables that together comprise a relational database of information necessary to install a group of applications.
installer function	The application programming interface called by an application to obtain installer services.
installer package authoring tool	A third-party tool that lets users create installation packages.
installer properties	Variables that are used during the installation.
INSTALLLEVEL	Initial level at which features are selected "ON" for installation by default.
integrated development environment (IDE)	A software program's interface. Also referred to as an interactive development environment.
interactive development environment (IDE)	A software program's interface. Also referred to as an integrated development environment.
interface	The point at which independent systems communicate with each other. The user interface of an application often consists of toolbars, menus, buttons, windows, and other items.
Internal Consistency Evaluator (ICE)	Tests that can be run on a Windows Installer database (using Orca, Msival2, or the Premier or Professional editions of InstallShield) to warn about potential (or actual) authoring errors.
internal source files	Files that are included in the installation archive file.
internal user interface	Built-in capabilities of an installer that can be used to create a graphical user interface that is displayed to the end user during installation.
internet Explorer (IE)	The browser developed by Microsoft and distributed with their Windows operating systems.

Table -1 • InstallShield Glossary

Term	Definition
ISM	See .ism file.
isolated application	An application that has been modified so that it always loads the versions of components, such as **.dll** files, with which it was originally developed and tested.
ISScript.msi	InstallScript engine installer. **ISScript.msi** installs the required files to run an installation. The InstallScript engine is also known as the InstallShield Scripting Run Time. InstallScript is a programming language used to create installations.
JAR	Acronym for Java archive file. See also .jar file.
Java Archive	See .jar file.
Java Native Interface (JNI)	A Java specification that enables Java code to call native code in a shared library. Native code in the JNI library can also invoke Java code through JNI.
Java Virtual Machine (JVM)	A simulated computer that interprets Java programs compiled into bytecode. The Java programs are usually stored in **.class** files.
JNI	See Java Native Interface.
JPEG	See .jpg file.
JVM	See Java Virtual Machine.
KB	See knowledge base.
key file	A unique file for each component that Windows Installer uses to detect the component's presence. In order to create advanced component settings or shortcuts, a key file must be specified.
key path	A unique registry value for each component that Windows Installer uses to detect the component's presence. A component can have either a key file or a key path, but not both.
knowledge base (KB)	A collection of technical information, instructions, and articles that are beyond the scope of the software's help information. The Knowledge Base features tips, tricks, and techniques, answers to Frequently Asked Questions, and articles on both technical and design issues. Flexera Software posts periodic updates to the Knowledge Base Articles on its Web site (http://www.installshield.com).
left-click	The act of selecting the left button on a mouse.
library	A collection of similar objects that are stored for occasional use, such as programs in source code or object code form, data files, scripts, and templates. A program library is a collection of (usually) precompiled, reusable programming routines that a programmer can "call" when writing code.

Table -1 • InstallShield Glossary

Term	Definition
load-ordering group	One service may require another service to already be running before it starts. For this reason, services need to be grouped and set to load in a specific order. Service load-ordering groups are listed under HKEY_LOCAL_MACHINE\System\CurrentControl Set\Control\ServiceGroupOrder. The service's Start Type property determines when it loads within its group.
localization	The process of adapting a product or service to a particular language and culture.
log database (LogDB)	A database that is on the target machine and contains a log of everything that was configured to be logged during installation and uninstallation.
log file	File containing a record of the activity that occurred during a software process (such as an installation, a build, a download) on a computer workstation or Web server. For example, Web servers maintain log files listing every request made to the server, summarizing files that were copied; bytes that were transferred; or which pages, images, and files are requested. A build log file lists all features, setup types, merge modules, dynamic links, and files included in the build.
LogDB	See log database.
logging	The process of recording the activity of a software process (such as an installation, a build, or a download) in a log file.
macro	An instruction that represents a sequence of instructions.
maintenance mode	When a user runs an installation program a second (or later) time for a product already installed on their system, the installation runs in maintenance mode. Maintenance mode enables the user to modify feature selections from the first-time installation, repair the features already installed, or remove the entire program.
major upgrade	A comprehensive update of the product that warrants a change in the ProductCode property. A new product code is required if you want to have two versions of a product installed on the same machine.
managed application	An application is called a "managed application" if elevated (system) privileges are used to install the application.
MD	See media descriptor.
media descriptor (MD)	The semantics of an installable unit (IU). It provides a mapping of the elements in the deployment descriptor (DD) to some physical source for the data, which is usually files.
merge module (MM)	A package containing all of the logic and files needed to install distinct pieces of functionality such as run-time **.dll** files and virtual machines. Merge modules are built once and can be added to any installation project.

Table -1 • InstallShield Glossary

Term	Definition
metafile	An image saved using the standard Windows metafile format. A metafile is a set of drawing instructions. Windows metafiles have a **.wmf** file extension.
MFC	See Microsoft Foundation Classes.
Microsoft Data Engine (MSDE)	A fully SQL Server–compatible data engine for building desktop and shared solutions. It provides an easy migration path to SQL Server 7.0. Solutions built with MSDE can be migrated to full SQL Server 7.0 without requiring a change in a single line of code.
Microsoft Developer Network (MSDN)	A set of online and offline services designed to help developers write applications using Microsoft products and technologies.
Microsoft disc operating system (MS-DOS)	The first operating system created by Microsoft. MS-DOS is the underlying operating system of Windows 95, 98, and ME. Windows NT, 2000, and XP operating systems support existing DOS applications.
Microsoft Foundation Classes (MFC)	A huge set of C++ classes that developers can use to write Win32 applications. To run an application that uses MFC, the MFC engine has to be installed on a target system.
MIDI	See musical instrument digital interface.
migrating	The process of moving from the use of one operating environment to another that is, in most cases, thought to be a better one. For example, moving from Windows NT Server to Windows 2000 Server would be considered a migration because it involves ensuring that new features are exploited and that old settings do not require changing; it also involves taking steps to ensure that current applications continue to work in the new environment. Migration could also mean moving from Windows NT to a UNIX-based operating system (or the reverse). You can also migrate data from one kind of database to another kind of database. This usually requires converting the data into some common format that can be output from the old database and input into the new database. Migration is also used to refer simply to the process of moving data from one storage device to another.
MIME	See multipurpose internet mail extensions.
minor upgrade	An update to a product in which the changes made to the installation do not warrant a change in the Product Code.
MM	See merge module.
modal	Requires the user to interact with it before the application can continue, restricting the user's activities with other windows or dialog boxes. See also modeless.
modeless	Enables the user to interact with other windows and dialog boxes. See also modal.
MSDE	See Microsoft Data Engine.

Table -1 • InstallShield Glossary

Term	Definition
MSDE named instance	A named instance of an installation of the Microsoft data engine (MSDE). See also named instance.
MSDE object template	A base for the MSDE merge module. When you add the MSDE merge module to a Windows Installer project or an InstallScript project, the InstallShield application starts with a base file (or stub) for this merge module. You need to configure this merge module, and then the application can apply your settings to the base file.
MSDN	See Microsoft Developer Network.
MS-DOS	See Microsoft disc operating system.
MS-DOS prompt	The Microsoft Disc-Operating System (MS-DOS) prompt is the visual indicator in the Command Prompt window signaling that MS-DOS is ready to accept a new command. The default MS-DOS prompt is `C:>`, followed by a blinking cursor.
MSI	See .msi file.
MSM	See .msm file.
MSP	See .msp file.
MST	See .mst file.
multimedia	A collection of various media such as sound, video, graphics, and animation used together to convey a message.
multipurpose Internet mail extensions (MIME)	The standard used by Web servers to identify the files that they are sending to Web clients. The MIME standard is a way of specifying both the type of file being sent and the method that should be used to turn it back into its original form.
musical instrument digital interface (MIDI)	A protocol designed for recording and playing back music on digital synthesizers. Unlike **.wav** files, which are digital recordings of actual sound (voice or music), MIDI files simply define the instruments and notes that are to be played, and how they should be played. A synthesizer (generally part of a MIDI-capable sound board) on the end-user's system reads the MIDI instructions and plays the music. Since MIDI files contain only instructions and not the actual music, they are many times smaller than digital audio files of the same duration. MIDI files also use less processor power to play.
named instance	An instance is a complete and independent installation of SQL Server on a given server. One default instance and any number (up to 16) of named instances can be installed on a single server. Apart from the management tools and client connectivity components that are shared between instances, each instance is effectively standalone. Each instance has its own security, can be started and stopped independently, and can even be service packed independently of other instances on the same server.

Table -1 • InstallShield Glossary

Term	Definition
nested installation	A type of custom action that installs or removes another installation package (sometimes called the child product) from within a running installation (called the parent product).
network	A network is two or more computers connected together to share hardware, software, and information. During installation, it may be necessary to move installation files that are on a network to your computer's hard drive to install the software properly.
NT service	An application that is installed on Windows NT to run as a service, meaning that once it is installed, execution is automatic and transparent to any user, and a user does not have to log in for the service to start. NT manages services that are defined and described in the registry.
object	A package containing all of the logic and files needed to install distinct pieces of functionality.
One Really Cool Application (ORCA)	An external tool for editing Windows Installer files.
One-Click Install	With a One-Click Install installation, the end user can download an application with minimal effort and can begin using the application immediately. The installation is automatically downloaded, uncompressed, and then executed automatically, with minimal user input. End users are not asked to specify a download location, and they are not required to manually launch the installation.
operating system (OS)	The software that controls the operation of a computer and directs the processing of programs by assigning storage space in memory and controlling input and output functions.
ORCA	See One Really Cool Application.
OS	See operating system.
package	All of the files needed to run an installation, including the installation database file, your application's files (separate from the installation database file or compressed into it), and an executable file (which may have all of the above files compressed inside it).
package code	The globally unique identifier for an installation package. See also globally unique identifier (GUID).
panel	A window within a wizard that contains one or more related settings that an end user can configure. See also wizard.
parent/child installation	See nested installation.

Table -1 • InstallShield Glossary

Term	Definition
patch	A special type of installation package that contains just the bits and portions of the application that is necessary to either update the application's files and installation to a specific version or to fix a bug in an earlier version.
patch file	A patch package used for patching. A patch package (**.msp**) file contains the transforms and instructions necessary for upgrading one or more installed versions of a product. Windows Installer uses a patch package to patch local or administrative installations. A patch package does not include a database like a regular installation package. Instead it contains at minimum one database transform that adds patching information to the database of its target installation package. The installer uses this information to apply the patch files that are stored in the cabinet file stream of the patch package.
patching	The method of updating an installation that replaces only the bits being changed rather than the entire application. This means that end users can download a patch for a product that is much smaller than the entire product.
path	In a computer operating system, a path is the route through a file system to a particular file. A path name is the specification of that path. Each operating system has its own format for specifying a path name. The DOS, Windows, and OS/2 operating systems use the following format: `driveletter:directorynamesubdirectorynamefilename.suffix`. UNIX-based systems use the following format: `/directory/subdirectory/filename`.
path variable	A variable that represents a location that can be defined once in a central location so that it is not necessary to change every source file's path each time that the project is moved or the folder structure is changed. You can instead use path variables to define commonly used paths once, and they are used during the development of your installation project. These paths do not apply to the target machines where the application is being installed. Rather, they are used to link to source files that need to be included in your installation project. When the project is built, those links are evaluated and the files they point to are built into the installation package.
PKCS	See public-key cryptography standards.
platform	The operating system for which the installation program is intended. An installation technology may be limited to creating installations for a specific platform(s).
preview mode	Mode for viewing the design of the user interface, or the current appearance of dialog and billboards. *Preview mode* is a term used in Windows Installer.
product	The actual application to be installed or uninstalled.
product code	A string that uniquely identifies a product.
progress bar	The visual indication of the progress of an executable file.
property	A value that has been placed on an object within an installation project that is used during installation or uninstallation by the installer or uninstaller.

Table -1 • InstallShield Glossary

Term	Definition
public property	A global variable whose value is set by the end user or system administrator. A public property can be set or changed during installation through interaction with the user interface as well as by setting the property on the command line, by applying a transform, or by using a standard or custom action. Unlike private property values, the values of public properties can be changed. Public property names cannot contain lowercase letters.
public-key cryptography standards (PKCS)	Developed by RSA Security, Inc., these specifications standardize aspects of public-key cryptography that are not covered by existing standards bodies.
publishing	A type of advertising ("just-in-time" installation) in which no user-interface elements are created for the component during installation, but the component can still be installed through Add and Remove Programs of the Control Panel or when an installed component requests the published component from the installer.
qualified component	A method of single-level indirection that is similar to a pointer. Qualified components are primarily used to group components with parallel functionality into categories.
QuickPatch	A special type of patch package that contains just the bits and portions of the database necessary to update your application's files and installation to a specific version. You can create a QuickPatch in InstallShield by using the QuickPatch type of project.
readme file	A text file that is included with the distribution of an application that contains important information. This information often pertains to the installation, uninstallation, or functionality that may not have been included in other product documentation.
red green blue (RGB)	The three colors of light that can be mixed to produce any other color. Colored images are often stored as a sequence of RGB triplets or as separate red, green, and blue overlays, though these are not the only possible representations. These three colors correspond to the three "guns" in a color cathode ray tube (CRT) and to the color receptors in the human eye.
redistributable	A merge module, object, or other file that may be legally distributed with an installation or application.
reduced user interface (UI)	The reduced UI level displays authored modeless dialogs, built-in modal error messages, and disc-prompt messages during installation. This UI level does not display any authored modal dialogs. This UI level uses the installer's internal user interface capabilities. See also modal, modeless, and internal user interface.
reference count	A tally that is incremented each time a shared file is installed. It is maintained under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ Windows\CurrentVersion\SharedDLLs.

Table -1 • InstallShield Glossary

Term	Definition
refresh build	This type of build only recompiles custom code. It can also perform single file replacement, meaning it can only rebuild and add or modify a single changed file without rebuilding the entire installation. This type of build is available for InstallScript projects only.
Regedt32.exe, Regedit.exe	The two versions of the Windows Registry Editor. The Registry Editor enables you to edit the entries in the registry. Regedt32 provides more functionality for editing the registry. If errors occur during installation, you may need to edit the registry to complete installation. **Regedt32.exe** and **Regedit.exe** have been merged in Windows XP computers, thus these two applications perform the same functionality in the new operating systems.
registry	A central database used by the Windows operating system to track the personal settings and the software and hardware installed on a computer. During installation, installation choices are written to the registry.
registry file	A text file of a predefined format that contains keys and values that can be merged into a registry.
reinstallation	When a product has been installed on a machine and its installation is run again, the installation reinstalls the product by overwriting its existing files, shortcuts, and registry entries.
relative path	A path that includes all of the information necessary to locate a file by starting at the current folder on the current drive, for example, **InstallShield\Support**. That folder can be located along that relative path only if it exists in the current directory.
release build	A full rebuild fit for releasing.
release notes	A file that is included with the distribution of an application. This file contains important information about the installation and uninstallation of the application. It can also contain information that may not have been included in other product documentation.
remote procedure call (RPC) stub	A small routine placed in a program that is used to request a service on another computer. RPC is a protocol, or agreed-upon format for transmitting data, that allows a program to request a service from a program located in another computer. The stub accepts the request from the program and forwards it to the remote procedure. When the procedure is complete, the stub receives the result and passes it back to the program that made the request.
repair mode	In repair mode, the installation checks for any type of corruption, such as missing or damaged files, incorrect registry entries, and self-registering files. If the installation detects any corruption, it attempts to repair the problem. See also maintenance mode.
resiliency	The ability for the application to reinstall components as necessary. If a component is accidentally deleted or corrupted, Windows Installer technology enables the application to essentially repair itself.

Table -1 • InstallShield Glossary

Term	Definition
restricted public property	A global variable for which the installation author can limit the ability to set or change it. Usually, only system administrators can manipulate restricted public properties. Restrictions are used to maintain a secure environment.
result set	The set of rows created by executing a **SELECT** statement.
RGB	See red green blue.
rich text format (RTF)	A Microsoft file format that contains special commands to indicate formatting information such as fonts and margins. RTF lets you exchange files between different word processors and operating systems.
rollback	The installation keeps track of all changes that are made during the installation process so that, if an error occurs and the installation is aborted, the changes will be "rolled back," that is, the machine will be restored to its original state.
row	A set of related columns that describe a specific entity, also known as a *record*.
RPC stub	See remote procedure call stub.
RTF	See rich text format.
run time	The time during which the installation interacts with the installer to install or uninstall your application on the target machine.
safe mode	Safe mode is a troubleshooting mode available in Windows 95, 98, ME, and 2000. When you start your computer in safe mode, only the operating system and mouse, keyboard, and display drivers are loaded. You may be able to start your computer in safe mode when it otherwise would not start at all. Safe mode lets you troubleshoot the operating system to determine what is not functioning properly.
schema	A description of the current structure of tables and views in a data source. The schema describes what columns are in each table, the data type of each column, and the relationships between tables.
SCM	See Service Control Manager.
SDK	See Software Development Kit.
Section 508	A United States law that amended the Rehabilitation Act to require federal agencies to make their electronic and information technology accessible to people with disabilities. Section 508 was enacted to eliminate barriers in information technology, to make available new opportunities for people with disabilities, and to encourage development of technologies that will help achieve these goals. The law applies to all federal agencies when they develop, procure, maintain, or use electronic and information technology. Under Section 508, agencies must give disabled employees and members of the public access to information that is comparable to the access available to others.

Table -1 • InstallShield Glossary

Term	Definition
security tool	A privacy tool that detects and eliminates destructive pests, such as Trojans, spyware, addware, and hacker tools on your computer. It complements anti-virus and firewall software, extending protection against non-viral malicious software that can evade the end user's existing security and invade privacy.
self-healing	See auto-repair.
self-registering file	A file that can enter information about itself in the Windows registry and remove that information upon uninstallation. Other types of files can be used without entering information into the registry. The installation of a self-registering file consists of installing the file to its desired location and then registering the file on your computer.
sequence	A collection of actions and dialogs that is executed sequentially in an installation project.
sequence tables	Tables that list the actions that control the installation process and specify their order of execution.
service	For Windows Installer–based or InstallScript-based projects, this is a program that runs in the background whenever a computer is running. Services perform tasks that do not require user interaction, such as software installation, process monitoring, file transfer, task scheduling, network management, and many more. In Win32, services are managed by the Service Control Manager.
Service Control Manager (SCM)	Maintains the system's database of services and exposes an interface for controlling these services.
service pack (SP)	An update to a software product that fixes existing problems and may provide product enhancements. The next version of a product incorporates all services packs previously released.
setup	See installation.
setup project	See installation project.
Setup.inx	The compiled script file. It is the object code that the installer engine executes.
shortcut	A file that points to an application. Clicking the shortcut is a fast way to open the application. Shortcuts are usually placed on the desktop or on the Start menu. See also advertised shortcut.
silent installation	An installation that is run without a user interface or any end user intervention.
small update	A patch that upgrades a package where both the installed package and the most recent one have the same version number.

Table -1 • InstallShield Glossary

Term	Definition
software development kit (SDK)	The documentation, samples, command-line compilers, debugging aids, utilities, and tools designed to enable developers to create applications and libraries that target a specific operating system. SDKs are usually provided by the manufacturer of the operating system.
source list	A list that specifies the locations where the installer searches for installation files. The entries in the source list can be network locations, URLs, or compact disks.
SP	See service pack.
splash screen	An image that is displayed to end users during the startup of a product installation. You can specify the image to be displayed, the length of time that it should display, and whether it contains localized images.
spyware	Any technology that aids in gathering information about a person or organization without their knowledge. On the Internet, spyware is programming that is put on someone's computer to secretly gather information about the user and relay it to advertisers or other interested parties. Spyware can be loaded on a computer as a software virus or as the result of installing a new program. Data collecting programs that are installed with the user's knowledge are not, properly speaking, spyware, if the user fully understands what data is being collected and with whom it is being shared.
SQL	See Structured Query Language.
SQL statement	A complete phrase in SQL that begins with a keyword and completely describes an action to be taken. For example, **SELECT * FROM Orders**.
standard action	An action that is built into an installation development software product. InstallShield products also support the creation and use of custom actions. See also action, custom action.
string table	A database that maintains the string IDs, values, and comments for all supported languages. These strings are used in dialogs and message boxes displayed to the end user at run time.
Structured Query Language (SQL)	A language used to work with databases.
summary information stream	The properties such as title, author, package code, templates, summary, and schema that are defined for a Windows Installer package and are used by the installer to install the application.
synchronous execution	The opposite of asynchronous execution, where control of the process is not released until the entire process has completed.
system policy	The rules and regulations by which a system must abide.

Table -1 • InstallShield Glossary

Term	Definition
system privileges	The system, programs, and functions available to each user.
table	A collection of rows of data.
TARGETDIR	In a Windows Installer–based installation, the TARGETDIR property specifies the root destination directory for the installation. In an InstallScript installation, the TARGETDIR system variable, by default, specifies the root destination directory for the installation.
Task Manager	A tool available on Windows platforms that provides information about the applications and processes running on your computer as well as your computer's performance. Prior to installation, you may need to use the Task Manager to end running applications and processes to prevent conflicts.
temp directory	A folder on your hard drive where the operating system or applications can temporarily store files while they are in use. When the application exits, the temporary files are deleted. It may be necessary to manually clean, or delete files from, the temp directory.
template	Something that establishes or serves as a pattern.
transform	A transform (**.mst**) file represents the differences between two installation databases. For example, network administrators may want to distribute different configurations of a product to the various departments in the company. As a result, you can create a transform for every configuration of the product, then apply the appropriate transform as needed.
transform error condition flags	A set of properties that are used to set the error conditions in a transform.
transform validation flags	The set of properties used to verify that a transform can be applied to the Windows Installer package.
tree node	A data structure that contains zero or more "child" nodes. Nodes are tree "roots". Tree nodes that do not have "children" are typically referred to as "leaves".
trees	Data that is structured like a tree.
UI	See user interface.
uncompress	See decompress.
uniform/universal resource locator (URL)	Represents the location of a Web site or page.
uninstallation	The undoing of an installation. Uninstallation is the installation maintenance option that enables the end user to remove the product files and reverse any changes that were made to the machine made during installation.

Table -1 • InstallShield Glossary

Term	Definition
uninstallation log file	A record of all uninstallation-related events that occurred during an installation. The log file is initialized at the beginning of the installation. If this log file becomes corrupted, it may result in an error during uninstallation.
unInstallShield	A component included in the Setup Wizard that uninstalls the product from the end user's PC.
unspanned CD/DVD media type	The payload is laid out in a CD/DVD archive, but no spanning across multiple media is used.
Update Manager (Windows or Java)	An optional Win32 or Java-based client application that can be used by end users to set their own update schedules, or to silence the update queries performed by the agent.
upgrade	A move from a lesser version of an application to a newer, usually improved version of an application.
upgrade code	The code required to install a newer version of existing software.
upgrading	The process of installing a newer version of existing software.
URL	See uniform/universal resource locator.
user interface (UI)	The user's access to the software.
user profile	Record of an individual end user's settings, such as shortcuts, favorites, and settings for application, display, and hardware. User profiles enable multiple users to share a single computer while maintaining their own preferences.
validation	The process of validating, that is, to confirm the true form of data.
value pack (VP)	An installation that upgrades the features and functions of an installed application.
variable	A value stored in the computer. Its value can be composed of any printable characters, numeric or text. Unlike a constant, whose value never changes, a variable's value can be changed at any time.
volume	The total amount of space, in blocks, on one piece of media.
VP	See value pack.
Web installation	An installation that end users can access and run from a Web site.
Web site	A location on the World Wide Web.
WinDir	Stores the path to the executable file for Win16 on Win32 (WOW).
window	The area in which an open application appears on a screen.

Table -1 • InstallShield Glossary

Term	Definition
Windows API	The Windows application programming interface (API), which provides the building blocks that are used by applications written for the Windows platform. Each API is a specific method prescribed by a computer operating system or by an application program. A programmer writing an application can make requests of the operating system or another application with the use of a Windows API. Each API has different system requirements to run properly.
Windows installation CD	The compact disc that contains the Windows operating system. If your computer manufacturer installed the operating system on your computer, the installation CD should have been included with your computer documentation. If you installed the Windows operating system on your computer, you used the installation CD to do so.
Windows Installer	(1) An installation and configuration service. It is based on a data-driven model and provides all installation data and instructions in a single, complete package. In the data-driven installation model, a master set of installation tables is created where every application resource (files, registry keys, and so on) is clearly tied to the component or feature it supports. The user selects the objects to install and where to install them, and the Windows Installer manages the procedural instructions. (2) Refers to the service, properties, and tables of an **.msi** package.
Windows Installer service	An operating system component that centrally manages application installation configuration and application uninstallation.
Windows NT service	See Windows service.
Windows service	Long-running executable applications that run in their own Windows sessions. For example, writing messages to an event log.
Windows system folder	Contains core operating system files, which are necessary to keep the computer running properly. Errors occurring during installation may be a result of missing or corrupt files contained in the System folder.
wizard	(1) A program utility that works as an interactive guide by walking the user step-by-step through an unfamiliar task. (2) The InstallShield Wizard is the wizard that displays during the run time of an installation or uninstallation. It presents the end user with a flexible, predefined series of dialogs that walk them through the installation or uninstallation process.
wizard dialog	Dialogs that display to the end user during installation and uninstallation. They enable end users to interact with the operation by reading or specifying information.
wizard interface	Used by the end user to interact with a wizard at run time.
WYSIWYG	What you see is what you get.

Table -1 • InstallShield Glossary

Term	Definition
XCopyFile	An InstallScript function that copies one or more files and subdirectories to a target directory, creating subdirectories on the target machine, if necessary.
XML	See extensible markup language.
XSL	See extensible stylesheet language.

Index

Symbols

_ISCRIPT_ISDEV preprocessor constant 793
    sample code 794
_ISCRIPT_ISPRO preprocessor constant 793
    sample code 794
_ISCRIPT_VER preprocessor constant 793
_serial_verifyCA_isx custom action 3201
_serial_verifyCA_isx_helper custom action 3201
.appx 1364
.cab files
    configuring maximum size for 319, 321
    creating .txt report about, for InstallScript installations 1680
    extracting files from, for InstallScript installations 1680
    from InstallScript installations, viewing 1677
    limitations 319, 321
    opening, from InstallScript installations 1679
    overview, from InstallScript installations 1678
.hdr files 1678
    creating a report about 1680
    from InstallScript installations, viewing 1677
    opening 1679
    overview 1678
.ilg 1681
    converting to .txt file 1683
    opening 1682
    overview 1681
    viewing 1681
.isproj 1189
.isv files 1725
.msi files
    converting to .ism files 270
    launching from the IDE 1232
    opening in direct mode 2032
    running multiple ones simultaneously 616
    troubleshooting conversion errors 2661
.msp files 2033
    creating 2505
    editing 2033
.mst files 1341
    applying 1343
    creating by comparing two .msi files 1342
    creating by starting with a single .msi package 1342
    customizing default responses for 1345
    editing 268
    modifying default responses for 1345
.NET 1728
    32-bit dependencies 537
    64-bit dependencies 537
    custom action 815
        32 bit vs. 64 bit 818
    properties and dependencies 535
    redistributables 635
.NET Framework
    version 2.0, 64-bit 634, 637
    version 3.0 634, 637
    version 3.0, 64-bit 634, 637
.pfx 1205
.prq files 2557
    editing 2557
    specifying an alternate URL for downloading 1434
.swf 1062
.vdproj 404
    converting or importing into an InstallShield project (.ism) 404
.xml 475
.zip 1363
DO_NOT_BUILD 399

Numerics

-10000 - Process Cancelled By User 2815
-10001 - Suite File Missing 2815
-10002 - Suite File is Duplicate 2816
-10003 - Application File Missing 2816
-10004 - INI File Missing 2816
11000 - Excluding TCPIP Registry Entries 2817
11001 - Fail on VMware 2817
11003 - Control Panel Applet - Citrix 2817
11004 - Control Panel Applet - ThinApp 2818
11005 - QuickTime 7.4.1 Causes Fatal Error 2818
11006 - Adobe Distiller Exclude AdobePDFSettings 2818
11007 - Exclude URL Shortcut 2819
27500 2665
27501 2665
27502 2665
27503 2666
27504 2666
27505 2666
27506 2666
27507 2666
27508 2667
27509 2667
27510 2667
27511 2667
27514 2668
27515 2668
27516 2668
27517 2668
27519 2669
27520 2669
27521 2669
27524 2669
27525 2670
27526 2670
27527 2670
27528 2670
27529 2670
27531 2670
27532 2670
27533 2670
27534 2671
27535 2671
27536 2671
27537 2671
27538 2671
27539 2671
27540 2671
27541 2671
27542 2672
27543 2672
27544 2672
27545 2672
27546 2672
27548 2672
27549 2672
27550 2673
27551 2673
27552 2673
27553 2673
27554 2673
27555 2674
32-bit
  build-time validation for 1168
  requirement for an InstallShield prerequisite 1898
404 error messages for a Web site, application, or virtual directory 1151
64-bit
  build-time validation for 1168
  components 241, 244
  operating systems, targeting 240
  operating systems, targeting with a Suite/Advanced UI or Advanced UI installation 245
  operating systems, targeting with a Windows Installer-based installation 241
  operating systems, targeting with an InstallScript installation 244
  registry reflection 571
  requirement for an InstallShield prerequisite 1898
  self-registration 241, 244
  support for connecting to an instance of Oracle 1108
  System32 folder source files 514
-9000 - Unknown Exception 2769
-9001 - Unknown COM 2769
-9002 - Error Opening Package 2769
-9003 - Error Saving Package 2770
-9004 - Process Cancelled By User 2770
-9005 - Error Creating Temporary Folder 2771
-9006 - Error Decompressing Package 2771
-9007 - File With Extension Not Found 2772
-9008 - Error Extracting Icon 2772
-9009 - Unknown Provider 2772
-9010 - Invalid Target File Name 2773
-9011 - Error Reading MSI Table 2773
-9012 - Unexpected Error in Method 2773
-9013 - Type Library Not Found 2774
-9014 - ShellExecute Failed 2774
-9015 - Unable to Determine Full Path for Driver 2774
-9016 - Contents of Table Ignored 2775
-9017 - .NET 1.x Assembly Not Supported 2776
-9018 - Custom Actions Warning 2776
-9019 - Conditionalized Components 2777
-9020 - Directory With Null Parent Error 2778
-9021 - Unable to Extract COM Data 2778
-9022 - Complus Table Error 2779
-9024 - FileSFPCatalog 2779
-9026 - LaunchCondition Table Warning 2779

-9027 - LockPermissions Table Warning 2780
-9028 - MoveFile Table Error 2781
-9029 - MsiDriverPackages Table Error 2781
-9030 - ODBCTranslator Table Warning 2782
-9031 - RemoveFile Table Warning 2782
-9032 - RemoveIniFile Table Warning 2783
-9033 - RemoveRegistry Table Warning 2783
-9036 - ISCEInstall Table Error 2784
-9037 - ISComPlusApplication Table Error 2784
-9038 - ISPalmApp Table Error 2785
-9039 - ISSQLScriptFile Table Error 2785
-9040 - ISVRoot Table Error 2786
-9041 - ISXmlFile Table Error 2786
-9051 - Package Decompression Canceled 2787
-9100 - CreateInstance of Package Object Failed 2787
-9101 - Create Operation of Package Object Failed 2787
-9102 - Failed to Write Header Information 2788
-9103 - Citrix Finalization Failed 2788
-9104 - Citrix Save Failed 2789
-9105 - Error Initializing Citrix Writer 2789
-9106 - Error Initializing Citrix Package 2789
-9107 - Error Writing Citrix File Entries 2790
-9108 - Error Determining Source File Path 2790
-9109 - Error Writing Citrix Folder Entries 2790
-9110 - Error Writing Citrix Registry Entries 2791
-9113 - Error Writing Citrix INI File Entries 2791
-9114 - Error Writing Citrix Shortcuts 2791
-9115 - Error Saving Citrix Profile 2792
-9116 - Error Creating Empty Citrix Profile 2792
-9117 - Error Creating Intermediate Folder 2792
-9118 - Error Initializing Citrix Profile 2793
-9119 - Error Creating Default Target in Citrix Profile 2793
-9120 - Error Deleting File From Profile 2793
-9121 - Failed to Copy File into Citrix Profile 2794
-9122 - Target Does Not Exist in Citrix Profile 2794
-9124 - No Shortcuts Created for this Profile 2794
-9125 - Error Writing Citrix File Type Associations 2795
-9126 - Failed to Sign Profile Using Certificate 2795
-9127 - Could Not Create Script Execution 2795
-9128 - Duplicate Shortcut 2796
-9129 - Duplicate Shortcut Names 2796
-9130 - Duplicate Shortcut Targets 2797
-9131 - Unable to Resolve Installer Variable 2797
-9132 - 16 Color Shortcut Icon Not Found 2797
-9133 - Shortcut Icon Not Found 2798
-9134 - Failure to Extract Icon from Executable 2798
-9135 - Shortcut Target is 16-Bit 2799
-9136 - Some Files May Not Be Decompressed 2799
-9137 - Destination Directory Cannot Be Found 2799
-9138 - DuplicateFile table warning 2800
-9150 warning 2801
-9151 error 2801
-9200 - ThinApp Must Be Installed 2801
-9201 - Extension for Shortcut Files Must Be .exe 2802
-9202 - No Applications Were Created 2802
-9203 - ThinApp Tool is Missing 2803
-9204 - Duplicate Shortcut 2803
-9205 - Identically-Named Shortcut Already Exists, But With Different Command Line Parameters 2803
-9206 - Identically-Named Shortcut Already Exists But With a Different Target 2804
-9207 - Error During Build Process (vregtool.exe) 2804
-9208 - Error Occurred During Build Process (vftool.exe) 2804
-9209 - Error Occurred During Build Process (tlink.exe) 2805
-9300 - Unhandled Exception During AdviseFile Operation 2805
-9301 - Unhandled Exception During AdviseRegistry Operation 2806
-9302 - Unhandled Exception During Command Action 2806
-9303 - Unhandled Exception During Alter File Action 2806
-9304 - Unhandled Exception During Alter Registry Action 2807
-9305 - Unhandled Exception During Create Action 2807
-9306 - Unhandled Exception During Execution of Rules Engine 2807
-9401 - Error Initializing App-V Writer 2808
-9402 - Error Initializing App-V Package 2808
-9403 - Error Writing App-V File Entries 2808
-9404 - Error Writing App-V Folder Entries 2809
-9405 - Error Writing App-V Registry Entries 2809
-9406 - Error Writing App-V INI File Entries 2809
-9407 - Error Writing App-V Shortcuts 2810
-9408 - Error Writing App-V File Type Data 2810
-9409 - Error Saving App-V Data 2810
-9410 - Error Determining Source File Path 2811
-9411 - OSD File Template Could Not Be Extracted 2811
-9412 - OSD File Could Not Be Saved 2811
-9413 - App-V OSD Real Save 2812
-9414 - Local App-V Application Should Not Be Specified as a Dependency of the Primary Application 2812
-9415 - Dependency Application Was Not Found 2812
-9416 - Invalid Primary Application Directory 2813
-9417 - Dependency Application's OSD File Contains an Invalid HREF Value 2813
-9418 - Error While Privatizing Side-By-Side Assemblies 2814
-9419 - Error Inserting Watermark 2814
-9424 - error building an App-V 5.x package 2814
-9500 - Shortcut Missing 2815

A

Action text 2367
  settings 2384, 2394
  specifying 834
Actions 2367
actions in a Suite/Advanced UI installation 1382
  .exe 1383
  DLL 1384

PowerShell 1387
scheduling 1396
types 1398
AddAdvancedFile Method 2837
AddComponent method 2839
AddComponentSubFolder Method 2889
AddCondition method 2914
AddCondition method (Advanced UI, Suite/Advanced UI) 3102, 3124
AddCustomAction method 2840
AddDetectCondition method (Advanced UI, Suite/Advanced UI) 3159
AddDynamicFileLink method (Advanced UI, Suite/Advanced UI) 3130
AddDynamicFileLinking method 2890
AddEligibleCondition method (Advanced UI, Suite/Advanced UI) 3159
AddEnvironmentVar method 2891
AddExitCondition method (Advanced UI, Suite/Advanced UI) 3060
AddExtensionCondition method (Advanced UI, Suite/Advanced UI) 3105
AddFeature method 2840
AddFile method 2891
AddFile method (Advanced UI, Suite/Advanced UI) 3131
AddFilter method (Advanced UI, Suite/Advanced UI) 3116
AddFolder method (Advanced UI, Suite/Advanced UI) 3132
AddGroup method (Advanced UI, Suite/Advanced UI) 3106
Adding 1345
file to source control 1726
InstallShield prerequisites to redistributables gallery 609
InstallShield prerequisites, merge modules, and objects to InstallScript projects 614
merge modules to redistributables gallery 611
prerequisites, merge modules, and objects to Windows Installer-based projects 612
Project Assistant dialog support to existing InstallShield Professional projects 397
server locations 1346
adding
adding files to an IIS virtual directory 1138
Additional Tools view 2521
AddLanguage method 2842
AddLanguage method (Advanced UI, Suite/Advanced UI) 3061
AddMergeModule method 2915
AddObject Method 2915
AddParameter method (Advanced UI, Suite/Advanced UI) 3107
AddPathVariable method 2842
AddPathVariable method (Advanced UI, Suite/Advanced UI) 3062
AddProductConfig method 2843
AddProperty method (Advanced UI, Suite/Advanced UI) 3062
AddRelease method 2938
AddRemoveFile method 2892
AddSetupFile Method 2845
AddSetupFile method (Advanced UI, Suite/Advanced UI) 3063
AddSetupType Method 2845
AddShellProperty method 2995
AddShortcut method 2926
AddStringEntry method 2930
AddStringEntry method (Advanced UI, Suite/Advanced UI) 3084
AddSubFolder method 2927
AddSuiteAction method (Advanced UI, Suite/Advanced UI) 3064
AddSuiteActionRef method (Advanced UI, Suite/Advanced UI) 3118
AddSuiteFeature method (Advanced UI, Suite/Advanced UI) 3065
AddSuitePackage method (Advanced UI, Suite/Advanced UI) 3066, 3160
AddSuiteRelease method (Advanced UI, Suite/Advanced UI) 3069
AddSuiteSubFeature method (Advanced UI, Suite/Advanced UI) 3124
AddSuiteTransaction method (Advanced UI, Suite/Advanced UI) 3069
Administration sequence 852
administrative privileges 1687
Advanced component settings 575
Advanced UI installation 261
Add or Remove Programs entry for 1402
adding .appx packages to 1364
adding .exe packages to 1361, 1362
adding .msi, .msp, and transaction packages to 1356
adding an InstallScript package to 1358, 1359
adding Web Deploy packages to 1363
automatic updates for 1418
custom package folder names 1378
dependency packages 1370
differences with Suite/Advanced UI installations 1351
downloadable updates for 1418
dynamic file linking 1365
feature state 1408
feature-package associations 1373
files included in 1349
formatted expressions 1410
guidelines for adding packages to 1355
importing .prq files into 1369
importing InstallShield prerequistes into 1369
installation order of packages 1371
InstallScript events, functions, and variables for 1358, 1359
logging a 1423
logging packages in a 1427
mode condition 1401
object expressions 1412, 3191

package operations for 1376
passing command-line parameters to packages in a 1403
password protecting 1422
primary packages 1370
properties 1709
requirements for 239
run-time locations of packages 1375
Setup.exe command-line parameters 3252
shared packages 1373
system searches in 1412
troubleshooting issues with 1423
UAC prompts 1413
unexpected maintenance mode 1423
Advertisement sequence 850
Advertising 595
features 595
publishing components 584
shortcuts 2259
ALLUSERS 1691
ampersands in the wizard interface 1034
ANSI 409
AppID 1946
Application Data view 2241
Application lifecycle 255
Application paths key 582
Application pools 1140
Applying transforms 1343
Appx support 1364
architecture validation 1168
ARPREADME 497
Assemblies 1558
patching in the global assembly cache 1558
AttachComponent method 2916
AttachPackage method (Advanced UI, Suite/Advanced UI) 3125
auto completion in a script editor 307
Automatic end-user responses for transforms 1345
automatic update notification 1077
adding a dialog to check for updates 1080
creating a shortcut to check for updates 1079
disabling in a project 1078
FlexNet Connect 2150
installation files needed for 1079
registering your application for 1081
Automation interface 2821
64-bit machines and 1746
AddAdvancedFile Method 2837
AddComponent method 2839
AddComponentSubFolder Method 2889
AddCondition method (Advanced UI, Suite/Advanced UI) 3102, 3124
AddCustomAction method 2840
AddDetectCondition method (Advanced UI, Suite/Advanced UI) 3159
AddDynamicFileLink method (Advanced UI, Suite/Advanced UI) 3130
AddDynamicFileLinking method 2890
AddEligibleCondition method (Advanced UI, Suite/Advanced UI) 3159
AddEnvironmentVar method 2891
AddExitCondition method (Advanced UI, Suite/Advanced UI) 3060
AddExtensionCondition method (Advanced UI, Suite/Advanced UI) 3105
AddFeature method 2840
AddFile method 2891
AddFile method (Advanced UI, Suite/Advanced UI) 3131
AddFilter method (Advanced UI, Suite/Advanced UI) 3116
AddFolder method (Advanced UI, Suite/Advanced UI) 3132
AddGroup method (Advanced UI, Suite/Advanced UI) 3106
AddLanguage method (Advanced UI, Suite/Advanced UI) 3061
AddMergeModule method 2915
AddObject Method 2915
AddParameter method (Advanced UI, Suite/Advanced UI) 3107
AddPathVariable method 2842
AddPathVariable method (Advanced UI, Suite/Advanced UI) 3062
AddProductConfig method 2843
AddProperty method (Advanced UI, Suite/Advanced UI) 3062
AddRelease method 2938
AddRemoveFile method 2892
AddSetupFile Method 2845
AddSetupFile method (Advanced UI, Suite/Advanced UI) 3063
AddSetupType Method 2845
AddShellProperty method 2995
AddShortcut method 2926
AddStringEntry method 2930
AddStringEntry method (Advanced UI, Suite/Advanced UI) 3084
AddSubFolder method 2927
AddSuiteAction method (Advanced UI, Suite/Advanced UI) 3064
AddSuiteActionRef method (Advanced UI, Suite/Advanced UI) 3118
AddSuiteFeature method (Advanced UI, Suite/Advanced UI) 3065
AddSuitePackage method (Advanced UI, Suite/Advanced UI) 3066, 3160
AddSuiteRelease method (Advanced UI, Suite/Advanced UI) 3069
AddSuiteSubFeature method (Advanced UI, Suite/Advanced UI) 3124

AddSuiteTransaction method (Advanced UI, Suite/Advanced UI) 3069
AddUpgradeTableEntry method 2847
AttachComponent method 2916
AttachPackage method (Advanced UI, Suite/Advanced UI) 3125
Build method 2982
Build method (Advanced UI, Suite/Advanced UI) 3175
build status events 1745
BuildPatchConfiguration method 2848
BuildPCPFile method 2850
CanMove method (Advanced UI, Suite/Advanced UI) 3113
ChangeGroupType method (Advanced UI, Suite/Advanced UI) 3108
CloseProject method 2851
CloseProject method (Advanced UI, Suite/Advanced UI) 3070
CreatePatch method 2851
CreateProject method 2852
CreateProject method (Advanced UI, Suite/Advanced UI) 3070
Delete method 3016
DeleteAdvancedFile Method 2853
DeleteCondition method (Advanced UI, Suite/Advanced UI) 3109, 3126, 3161
DeleteCustomAction method 2855
DeleteDynamicFileLink method (Advanced UI, Suite/Advanced UI) 3132
DeleteExitCondition method (Advanced UI, Suite/Advanced UI) 3071
DeleteFile Method 2903
DeleteFile method (Advanced UI, Suite/Advanced UI) 3133
DeleteFilter method (Advanced UI, Suite/Advanced UI) 3117
DeleteFolder method (Advanced UI, Suite/Advanced UI) 3134
DeleteLanguage method (Advanced UI, Suite/Advanced UI) 3072
DeleteMergeModule method 2855
DeleteParameter method (Advanced UI, Suite/Advanced UI) 3110
DeletePathVariable method (Advanced UI, Suite/Advanced UI) 3072
DeleteProperty method (Advanced UI, Suite/Advanced UI) 3073
DeleteRelease method 2939
DeleteSetupFile Method 2858
DeleteSetupFile method (Advanced UI, Suite/Advanced UI) 3074
DeleteSetupType Method 2859
DeleteShellProperty method 2996
DeleteShortcut method 2928
DeleteStringEntry method 2931
DeleteStringEntry method (Advanced UI, Suite/Advanced UI) 3085
DeleteSubFolder method 2929
DeleteSuiteAction method (Advanced UI, Suite/Advanced UI) 3075
DeleteSuiteActionRef method (Advanced UI, Suite/Advanced UI) 3119
DeleteSuiteFeature method (Advanced UI, Suite/Advanced UI) 3076
DeleteSuitePackage method (Advanced UI, Suite/Advanced UI) 3076
DeleteSuiteRelease method (Advanced UI, Suite/Advanced UI) 3077
DeleteSuiteSubFeature method (Advanced UI, Suite/Advanced UI) 3126
ExportProject method 2861
ExportStrings method 2862
ForceUpgrade method 2863
ForceUpgrade method (Advanced UI, Suite/Advanced UI) 3078
GenerateGUID method (Advanced UI, Suite/Advanced UI) 3078
ImportProject method 2864
ImportRegFile method 2896
ImportStrings method 2865
InsertCustomAction method 3038
ISWiAdvancedFile Object 2869
ISWiAdvancedFiles Collection 3016
ISWiAutomaticUpgradeEntry object 2870
ISWiComponent object 2872
ISWiComponents collection 3018
ISWiComponentSubFolder Object 2901
ISWiComponentSubFolders Collection 3020
ISWiCondition object 2903
ISWiConditions collection 3021
ISWiCustomAction object 2871
ISWiCustomActions collection 3022
ISWiDynamicFileLinking object 2904
ISWiDynamicFileLinkings collection 3023
ISWiEnvironmentVar object 2906
ISWiEnvironmentVars collection 3024
ISWiFeature object 2908
ISWiFeatures collection 3025
ISWiFile object 2919
ISWiFiles collection 3026
ISWiFolder object 2925
ISWiFolders collection 3027
ISWiLanguage object 2929, 3028
ISWiLanguage object (Advanced UI, Suite/Advanced UI) 3083
ISWiLanguages collection (Advanced UI, Suite/Advanced UI) 3176
ISWiObject Object 2931
ISWiObjects Collection 3029

ISWiPathVariable object 2932
ISWiPathVariable object (Advanced UI, Suite/Advanced UI) 3085
ISWiPathVariables collection (Advanced UI, Suite/Advanced UI) 3177
ISWiPathVariables object 3030
ISWiProductConfig object 2935
ISWiProductConfigs collection 3031
ISWiProject object 2822
ISWiProject object (Advanced UI, Suite/Advanced UI) 3053
ISWiProperties collection 3033
ISWiProperties collection (Advanced UI, Suite/Advanced UI) 3178
ISWiProperty object 2940
ISWiProperty object (Advanced UI, Suite/Advanced UI) 3089
ISWiRelease object 2941
ISWiReleases collection 3034
ISWiRemoveFileObject object 2985
ISWiRemoveFiles collection 3035
ISWiRemoveFiles method 2897
ISWiSequence collection 3036
ISWiSequenceRecord object 2986
ISWiSetupFile Object 2987
ISWiSetupFile object (Advanced UI, Suite/Advanced UI) 3090
ISWiSetupFiles Collection 3039
ISWiSetupFiles collection (Advanced UI, Suite/Advanced UI) 3179
ISWiSetupType Object 2988
ISWiSetupTypes Collection 3040
ISWiShellProperties collection 3041
ISWiShellProperty object 2989
ISWiShortcut object 2990
ISWiShortcuts collection 3042
ISWiSISProperties collection 3043
ISWiSISProperty object 2997
ISWiStringEntries collection 3050
ISWiStringEntry object 3014
ISWiSuiteAction object (Advanced UI, Suite/Advanced UI) 3091
ISWiSuiteActionRef object (Advanced UI, Suite/Advanced UI) 3098
ISWiSuiteActionRefs collection (Advanced UI, Suite/Advanced UI) 3179
ISWiSuiteActions collection (Advanced UI, Suite/Advanced UI) 3180
ISWiSuiteCondition object (Advanced UI, Suite/Advanced UI) 3100
ISWiSuiteConditionParameter object (Advanced UI, Suite/Advanced UI) 3111
ISWiSuiteConditionParameters collection (Advanced UI, Suite/Advanced UI) 3180
ISWiSuiteConditions collection (Advanced UI, Suite/Advanced UI) 3181
ISWiSuiteDFLFilter object (Advanced UI, Suite/Advanced UI) 3112
ISWiSuiteDFLFilters collection (Advanced UI, Suite/Advanced UI) 3182
ISWiSuiteDynamicFileLink object (Advanced UI, Suite/Advanced UI) 3115
ISWiSuiteDynamicFileLinks collection (Advanced UI, Suite/Advanced UI) 3183
ISWiSuiteEvent method (Advanced UI, Suite/Advanced UI) 3079
ISWiSuiteEvent object (Advanced UI, Suite/Advanced UI) 3118
ISWiSuiteEvents collection (Advanced UI, Suite/Advanced UI) 3183
ISWiSuiteExitCondition object (Advanced UI, Suite/Advanced UI) 3120
ISWiSuiteExitConditions collection (Advanced UI, Suite/Advanced UI) 3184
ISWiSuiteFeature object (Advanced UI, Suite/Advanced UI) 3121
ISWiSuiteFeatures collection (Advanced UI, Suite/Advanced UI) 3185
ISWiSuiteFile object (Advanced UI, Suite/Advanced UI) 3128
ISWiSuiteFolder object (Advanced UI, Suite/Advanced UI) 3129
ISWiSuiteFolders collection (Advanced UI, Suite/Advanced UI) 3187
ISWiSuiteOperation object (Advanced UI, Suite/Advanced UI) 3134
ISWiSuitePackage object (Advanced UI, Suite/Advanced UI) 3140
ISWiSuitePackages collection (Advanced UI, Suite/Advanced UI) 3188
ISWiSuiteRelease object (Advanced UI, Suite/Advanced UI) 3163
ISWiSuiteReleases collection (Advanced UI, Suite/Advanced UI) 3188
ISWiUpgradeTable object 3015
ISWiUpgradeTableEntries collection 3050
Move method (Advanced UI, Suite/Advanced UI) 3110
MoveDown method (Advanced UI, Suite/Advanced UI) 3114, 3162, 3163
MoveTo method (Advanced UI, Suite/Advanced UI) 3099
MoveUp method (Advanced UI, Suite/Advanced UI) 3115
OpenProject method 2866
OpenProject method (Advanced UI, Suite/Advanced UI) 3081
RemoveComponent Method 2917
RemoveComponentSubFolder Method 2898
RemoveDynamicFileLinking method 2899
RemoveEnvironmentVar method 2899
RemoveFeature method 2868

RemoveFile method 2900
RemoveMergeModule method 2918
RemoveObject Method 2919
RemovePackage method (Advanced UI, Suite/Advanced UI) 3127
RemoveRemoveFile method 2901
RemoveSequenceRecord method 3038
SaveProject method 2869
SaveProject method (Advanced UI, Suite/Advanced UI) 3082
Standalone Automation Interface 1188
Suite/UI and Advanced UI projects and 3053
AVI files
in InstallScript and InstallScript MSI projects
enabling 1073

B

Background window
displaying during InstallScript and InstallScript MSI installations 1073
Basic MSI setup projects 262
Batch build 580
BATCH_INSTALL
and behavior for installation and uninstallation 1319
Behavior and Logic view 2361
Behavior editor 901
Best practices 541
dynamic file linking 527
for creating an installation 541
billboards 1058
in Basic MSI projects 1058
adding Flash files for 1062
adding images for 1063
configuring settings 1063
previewing without building and running a release 1064
removing 1065
run-time behavior for 1064
screen shot samples 1059
setting the order of 1064
specifying which type to use 1062
supported file types 1058
types 1059
in InstallScript and InstallScript MSI projects 1066
changing placement 1072
code to implement 1071
enabling 1073
naming files 1068
removing 1072
screen shot samples 1066
setting the order of 1071
special effects 1072
types 1066
settings for Flash and image files 2411
view for adding and configuring 2410
Bit flags 771
Bitmap control 941
bookmarks in a script editor 306
brush 1021
applying a brush to a background in the wizard interface 1027
applying a brush to a control in the wizard interface 1026
defining a custom one for the wizard interface 1024
for the wizard interface 1021
build events 1181
Build method 2982
Build method (Advanced UI, Suite/Advanced UI) 3175
Building a release 419, 439, 2436
at the command line 1173
batch build 1180
building a self-extracting executable file from the command line 1179
canceling a build 1184
copying a release 1230, 1231
including Windows Installer 633
passing parameters in an .ini file 1174
quick build 1180
rebuild a release 1179
specifying the run-time location of prerequisites for 1203
troubleshooting build errors and warnings 2568
using the automation interface 2982

C

C7501 warning 2767
C7502 warning 2767
C7503 warning 2768
C7505 warning 2768
C8001 error 2726
C8002 error 2727
C8003 error 2727
C8004 error 2728
C8005 error 2728
C8006 error 2728
C8007 error 2729
C8008 error 2729
C8009 error 2729
C8010 error 2730
C8011 error 2730
C8012 error 2730
C8013 error 2731
C8014 error 2731
C8015 error 2731
C8016 error 2732
C8017 error 2732
C8018 error 2733
C8019 error 2733

C8020 error 2733
C8021 error 2734
C8022 error 2734
C8023 error 2734
C8024 error 2735
C8025 error 2735
C8026 error 2736
C8027 error 2736
C8028 error 2736
C8031 error 2737
C8032 error 2737
C8033 error 2737
C8034 error 2738
C8035 error 2738
C8036 error 2738
C8037 error 2739
C8038 error 2739
C8039 error 2739
C8040 error 2740
C8042 error 2740
C8043 error 2740
C8044 error 2741
C8045 error 2741
C8046 error 2741
C8047 error 2741
C8048 error 2742
C8049 error 2742
C8050 error 2742
C8051 error 2743
C8052 error 2743
C8053 error 2743
C8054 error 2743
C8055 error 2744
C8057 error 2744
C8058 error 2744
C8059 error 2745
C8062 error 2745
C8063 error 2745
C8064 error 2746
C8065 error 2746
C8066 error 2746
C8067 error 2746
C8068 error 2747
C8069 error 2747
C8070 error 2747
C8071 error 2748
C8072 error 2748
C8073 error 2748
C8074 error 2749
C8075 error 2749
C8076 error 2749
C8077 error 2750
C8078 error 2750
C8079 error 2750
C8080 error 2751
C8081 error 2751
C8082 error 2751
C8083 error 2752
C8084 error 2752
C8085 error 2752
C8086 error 2753
C8087 error 2753
C8088 error 2753
C8089 error 2753
C8090 error 2754
C8091 error 2754
C8092 error 2754
C8093 error 2755
C8097 error 2755
C8098 error 2755
C8099 error 2756
C8100 error 2756
C8101 error 2756
C8112 error 2757
C8113 error 2757
C8114 error 2757
C8115 error 2757
C8126 error 2758
C8127 error 2758
C8128 error 2758
C8522 error 2759
C9001 error 2766
Cabinet and Log File Viewer 1677
CanMove method (Advanced UI, Suite/Advanced UI) 3113
Chained .msi packages 1163
  overview 1163
  that are processed as a single transaction 1163
Chaining installations 616
ChangeGroupType method (Advanced UI, Suite/Advanced UI) 3108
Check box control 943
CheckForProductUpdates custom action 3201
CheckForProductUpdatesOnReboot custom action 3201
Checking a file into source control 1727
Checking a file out of source control 1727
CloseProject method 2851
CloseProject method (Advanced UI, Suite/Advanced UI) 3070
Code pages 409
  installing 410
  requirements 409
COM components 546
  Extracting COM registration at build time 567
  Extracting COM registration when files are added 534
  registry-free registration 652
  sample application manifest 654
COM extraction
  excluding registry changes from 648
  with or without administrative privileges 246

COM objects 786
COM registration 646
COM server 645
  registering 645
COM+ applications 2344
  adding to your project 2344
  application proxy support 1123
  conditionally installing for servers and proxies 1124
  server applications 1122
Combo box control 948
Command-line building 1173, 1179
  self-extracting executable file 1179
Companion files 532
Company name
  displayed on Setup.exe's Properties dialog box 1329
  displayed on Update.exe's Properties dialog box 1519
Compile.exe 3214
Compiler 3214
  for InstallScript 3214
Component services 2344
  adding to your project 2344
  application proxy support 1123
  conditionally installing for servers and proxies 1124
  server applications 1122
Component wizard
  Best Practices option 544
  component type option 547
Components 2167
  adding subfolders 555
  advanced settings 575
  app paths entries 582
  associating with features 550
  COM 546
  conditions 562
  creating 543
  creating for application proxy support 1123
  creating for server applications 1122
  creating registry entries 554
  creating shortcuts 554
  definition 254
  destination folder 558
  file types 579
  fonts 546
  installing shared files 563
  installing, controlling, and configuring Windows services 744
  making data permanent 561
  publishing 584
  remote installation 566
  self-registration 650
  specifying destination from the script 559
  via the automation layer 3018
condition group 1601
condition tree 1601
Conditional installation 479
Conditions 426, 448, 1591
  and components 562
  and features 590
  and InstallShield prerequisites 1435
  and products 479
  in Suite/Advanced UI and Advanced UI projects 1601
  syntax 1593
  tips on how to create them in Suite/Advanced UI and Advanced UI projects 1639
  types of checks in Suite/Advanced UI and Advanced UI projects 1604
Configurations, using one project to create setups for multiple product configurations 1212
context-sensitive help 250
Control events 902
control theme 1021
  for the wizard interface 1021
controls 935
  HTML and hyperlink support 892
Converting 272
  from one project type to another 272
  Visual Studio project to InstallShield project 404
copyright for Setup.exe 1329
copyright for Update.exe 1519
Cost 790
Create New QuickPatch Wizard 1968
CreatePatch method 2851
CreateProject method 2852
CreateProject method (Advanced UI, Suite/Advanced UI) 3070
Creating 543
  custom actions, guidelines for 806
  features 588
  InstallShield prerequisites 1433
  QuickPatch project 1534
    Create New QuickPatch Wizard 1968
    for an existing QuickPatch 1535
    overview 1534
  response transforms 1345
  setup projects 268
  transform by comparing two .msi files 1342
  transform by starting with a single .msi package 1342
Creating a Basic MSI project 432
Custom actions 798
  _serial_verifyCA_isx 3201
  _serial_verifyCA_isx_helper 3201
  32- vs. 64-bit 1168
  CheckForProductUpdates 3201
  CheckForProductUpdatesOnReboot 3201
  copying from one sequence to another 856
  creating 798
  deferred, commit, rollback types, and properties 809
  DLLWrapCleanup 3201
  DLLWrapStartup 3201

documenting behavior of 805
icons 2369
IISSQLServerRollbackLoginInfo 3209
including in QuickPatch 2552
inserting into sequences 855
Installscript custom actions 810
InstallShield, descriptions of 3201
ISComponentServiceCosting 3202
ISComponentServiceFinalize 3202
ISComponentServiceInstall 3202
ISComponentServiceRollback 3202
ISComponentServiceUninstall 3202
ISJITCompileActionAtInstall 3203
ISJITCompileActionAtUnInstall 3203
ISNetApiInstall 3203
ISNetApiRollback 3204
ISNetCreateIniForOneUser 3204
ISNetDeleteIniFile 3204
ISNetGetGroups 3204
ISNetGetServers 3204
ISNetGetUsers 3204
ISNetSetLogonName 3204
ISNetValidateLogonName 3204
ISNetValidateNewUserInformation 3204
ISNetworkSharesCosting 3205
ISNetworkSharesFinalize 3205
ISNetworkSharesInstall 3205
ISNetworkSharesRollback 3205
ISNetworkSharesUninstall 3205
ISPrint 3205
ISQuickPatchFinalize 3205
ISQuickPatchFixShortcut 3205
ISQuickPatchHelper 3206
ISQuickPatchInit 3206
ISQuickPatchInit9X 3206
ISQuickPatchInit9X2 3206
ISRunSetupTypeAddLocalEvent 3206
ISSelfRegisterCosting 3207
ISSelfRegisterFiles 3207
ISSelfRegisterFinalize 3207
ISSetAllUsers 3207
ISSetTARGETDIR 3207
ISSetupFilesCleanup 3207
ISSetupFilesExtract 3207
ISSQLQueryDatabases 3208
ISSQLServerCosting 3208
ISSQLServerFilteredList 3208
ISSQLServerFinalize 3208
ISSQLServerInitialize 3208
ISSQLServerInstall 3208
ISSQLServerList 3208
ISSQLServerRemoveLoginInfo 3208
ISSQLServerRollback 3208
ISSQLServerUninstall 3209
ISSQLServerValidate 3209
ISSQLServerWriteLoginInfo 3209
ISUnSelfRegisterFiles 3209
ISVerifyScriptingRuntime 3209
ISXmlAppSearch 3209
ISXmlCosting 3210
ISXmlFinalize 3210
ISXmlInstall 3210
ISXmlRollback 3210
ISXmlUnInstall 3210
LaunchProgramFileFromSetupCompleteSuccess 3210
LaunchReadmeFileFromSetupCompleteSuccess 3210
managed assembly 815
passing parameters to a DLL 814
running a PowerShell script 820
setAllUsersProfile2K 3210
SetAllUsersProfileNT 3210
SetARPINSTALLLOCATION 3210
setUserProfileNT 3211
ShowMsiLog 3211
specifying progress messages for 834
terminate a process 819
types 2369
Windows logo requirements 806
WiseScriptFinalize 3211
WiseScriptInitialize 3211
WiseScriptRollback 3211
Custom error messages for a Web site, application, or virtual directory 1151
Custom Setup dialog 910
Custom Transfer File Operations 790

D

Database (SQL) 1996
Debug InstallScript 756
Default 1345
language 1566
Deferred custom actions and properties 809
DeleteAdvancedFile Method 2853
DeleteComponent method 2854
DeleteCondition method 2917
DeleteCondition method (Advanced UI, Suite/Advanced UI) 3109, 3126, 3161
DeleteCustomAction method 2855
DeleteDynamicFileLink method (Advanced UI, Suite/Advanced UI) 3132
DeleteExitCondition method (Advanced UI, Suite/Advanced UI) 3071
DeleteFile Method 2903
DeleteFile method (Advanced UI, Suite/Advanced UI) 3133
DeleteFilter method (Advanced UI, Suite/Advanced UI) 3117
DeleteFolder method (Advanced UI, Suite/Advanced UI) 3134
DeleteLanguage method (Advanced UI, Suite/Advanced UI)

3072
DeleteMergeModule method 2855
DeleteParameter method (Advanced UI, Suite/Advanced UI) 3110
DeletePathVariable method 2856
DeletePathVariable method (Advanced UI, Suite/Advanced UI) 3072
DeleteProductConfig method 2857
DeleteProperty Method 2857
DeleteProperty method (Advanced UI, Suite/Advanced UI) 3073
DeleteRelease method 2939
DeleteSetupFile Method 2858
DeleteSetupFile method (Advanced UI, Suite/Advanced UI) 3074
DeleteSetupType Method 2859
DeleteShellProperty method 2996
DeleteShortcut method 2928
DeleteStringEntry method 2931
DeleteStringEntry method (Advanced UI, Suite/Advanced UI) 3085
DeleteSubFolder method 2929
DeleteSuiteAction method (Advanced UI, Suite/Advanced UI) 3075
DeleteSuiteActionRef method (Advanced UI, Suite/Advanced UI) 3119
DeleteSuiteFeature method (Advanced UI, Suite/Advanced UI) 3076
DeleteSuitePackage method (Advanced UI, Suite/Advanced UI) 3076
DeleteSuiteRelease method (Advanced UI, Suite/Advanced UI) 3077
DeleteSuiteSubFeature method (Advanced UI, Suite/Advanced UI) 3126
dependencies
  64-bit 642
  dynamic scanning 642
  reviewing scanning results 643
  static scanning 641
Dependency scanners 2523
  filtering files and 643
Deployment Image Servicing and Management 1379
DeselectLanguage method 2860
DetectIIS6AppPool32BitSupport 1146
DetectIIS6Interfaces 1149
detection conditions in Suite/Advanced UI and Advanced UI projects 1601
Device Driver Wizard 2003
Device Drivers
  InstallScript 793
Device drivers 583, 2004
  sequencing installation
    Device Driver Advanced Setting 583
  signing
    Device Driver Wizard - Device Driver Package Panel 2004
Device drivers (automation) 2892, 2894
Dialog box properties 952
Dialog controls 935
  HTML and hyperlink support 892
Dialog Editor 430, 450
Dialog Editor, Basic MSI projects 899
Dialog Editor, InstallScript projects 883
Dialog Sampler 895
Dialog skins 895
  limitations of 895
Dialog theme 912, 927
Dialog Wizard 2007
Dialogs 871
  adding one to your installation to check for updates 1080
  Dialog Sampler 895
  removing from a project 878
  run-time language support for 408
  view 871
Differences 1667
Differential versus full releases 1507
DIFx 583, 793
Digital certificate 1205
Digital signing 1205
  application 1205
  files 1205
  patch packages 1530
  QuickPatch packages 1539
  timestamp server 318
DIM 1485
  .ini file changes in 1491
  associating with a feature in a Basic MSI project 656
  benefits of 1485
  configuring SQL servers in 1492
  creating components in 1490
  default destination 1486
  identifying its elements in a Basic MSI project 661
  Import DIM Wizard 2021
  importing into a Basic MSI project (irreversible) 660
  including files and folders in 1490
  managing COM+ applications in 1492
  managing Internet Information Services in 1492
  methods for including in an installation project 655
  ODBC resource configuration in 1491
  overriding its destination 658
  overriding path variables in 661
  referencing in a Basic MSI project 655
  registry changes in 1491
  resolving build-time conflicts 657
  resolving design-time conflicts 660
  scheduling custom actions and dialogs in 659
  searching for installed data in 1491
  setting environment variables in 1491

shortcuts in 1491
text file changes in 1491
using custom actions in 1491
using path variables in 1486
using Windows Installer properties in 1491
viewing build instructions for 658
XML file changes in 1491
XML vs. binary format 1485
DIM References view 2238
Direct Edit Mode 267
Direct Editor 2524
column header schema tooltips 2528
table record reference tracking 2529
Direct MSI 267
Directory combo control 955
Directory list control 958
DirectX 9 object 639
DirectX Object Wizard 2011
DisableSharedComponent 572
Disk Spanning 1324
Custom Disk Spanning 1325
Disk Spanning Rules 1325
Disk1 Folder, adding files to 867
DISM.exe 1379
DLL functions
calling 777
passing arrays to 778
passing parameters 814
using .def files 812
DLL Hell 652
DllRegisterServer 650
DllUnregisterServer 650
DLLWrapCleanup custom action 3201
DLLWrapStartup custom action 3201
Downgrades, preventing 468
downloading latest redistributables to your computer 607
DPI-aware support and images in the InstallScript UI 881
DPI-aware support and images in the Suite/Advanced UI and Advanced UI installations 1048
Dynamic file linking 525
adding 529
best practices for component creation 527
by-directory method 527
in Advanced UI and Suite/Advanced UI projects 1365
limitations 526

E

Edit field control 960
elevated privileges 462
eligibility conditions in Suite/Advanced UI and Advanced UI projects 1601
Embedding Custom Transfer File Operations 790
Enable Maintenance project property 488
Enabling automatic update notification 1077
End-user dialogs 2403
behavior 901
check box properties 943
combo box properties 948
creating 899
customizing default responses for 1345
deleting 878
dialog properties 952
display at run time 898
edit field properties 960
editing the layout in the Dialog editor 899
exporting 874
importing 874
localizing 1574
push button properties 992
text area properties 1008
themes for 912, 927
End-User License Agreement
requiring end users to read 931
Environment variables 2289
adding and modifying 2289
in path variables 507
Errors
build 2568
DIFx 2719
media build 2654
Setup.exe run time (Basic MSI and InstallScript MSI) 2677
Setup.exe run time (InstallScript) 2674
Setup.exe run time (Suite/Advanced UI and Advanced UI) 2681
Errors, configuring for a Web site, application, or virtual directory 1151
Ether object 1217
EULA
requiring end users to read 931
EulaScrollWatcher.dll 931
Events view 2393
Executable 822
creating a single executable file for distribution 1197
including InstallShield prerequisites when 625
launching 822
Execute sequences 854
exit conditions in Suite/Advanced UI and Advanced UI projects 1601
Export 801
Export Components 2017
Exporting dialogs 874
ExportProject method 2861
ExportStrings method 2862
Expressing Large Numbers in InstallScript 792

F

F8501 error 2760
F8502 error 2761
F8503 error 2761
F8504 error 2761
F8505 error 2762
F8506 error 2762
F8508 error 2763
F8509 error 2763
F8510 error 2763
F8511 error 2764
F8512 error 2764
F8513 error 2764
F8514 error 2765
F8515 error 2765
F8516 error 2765
F8517 error 2765
F8518 error 2766
F8519 error 2766
feature conditions in Suite/Advanced UI and Advanced UI projects 1601
Features 2154
adding components 550
advertising 595
definition 254
destination folder 589
enabling Windows roles and 1379
Install Level property 596
installing conditionally 590
leaving files uncompressed in a disk image 1198
making required 594
ordering 598
release flags 1208
remote installation 596
specifying features and subfeatures in function calls 789
via the automation layer 3025
visibility (Display setting) 592
Features, creating 434
File Open dialog 928
File properties 1815
File and Directory Lock Permissions 533
File types 579
File versioning rules 531
FileBrowse custom action 928
Files 511
attributes 520
dynamically linking to source locations 525
finding 519
leaving uncompressed in a disk image 1198
overwriting on the target system 531
specifying destination from the script 559
specifying for prerequisites 1438
Files and Folders view 2242
Files, adding 435
Filters.xml 643
specifying dependency scanner exclusions 643
specifying IIS exclusions for importing IIS data 1131
specifying registry change exclusions for COM extraction 648
Flash file 1062
adding as a billboard in a Basic MSI project 1062
Flash files
in InstallScript and InstallScript MSI projects
enabling 1073
FlexNet Code Aware integration 67, 1754
FlexNet Connect 2150
adding a dialog to check for updates 1080
automatic update notification 1077
creating a shortcut to check for updates 1079
disabling in a project 1078
installation files needed for 1079
registering your application with 1081
Font components 546
Fonts, installing with InstallScript projects 539
ForceUpgrade method 2863
ForceUpgrade method (Advanced UI, Suite/Advanced UI) 3078
Function wizard 2020

G

General Information view 2112
settings 2113
GenerateGUID method (Advanced UI, Suite/Advanced UI) 3078
GetLastError method 1217
Global assembly cache 1558
patching assemblies in 1558
Globalization 451, 1565
adding languages 1576
and dialogs 1574
code pages 409
default language 1566
including languages in the release 1571
language identifiers 1576
letting the user choose the language 1575
selecting supported setup languages 1569
tips 1565
Group box control 964
GUIDs 273

H

help
context-sensitive 250
using 249
high/low values 792

History of a QuickPatch project 2552
HKEY_USER_SELECTABLE 697
HTML in InstallScript or InstallScript MSI dialogs 892
HTTP errors, configuring 1151
hyperlinks on Basic MSI dialogs 967
hyperlinks on InstallScript or InstallScript MSI dialogs 892
Hyper-V 1598

I

ICEs 1256
Icon control 970
Icons
  for shortcuts 667
  specifying for Setup.exe 1334
  specifying for Update.exe 1520
IDE reference
  IIS virtual directories 1125
IIS 1125
  application mappings 1148
  application pool 1140
  ASP.NET platform 1146
  ASP.NET version 1145
  configuring error messages for 1151
  Enable32BitAppOnWin64 1146
  error -2147467259 1125
  feature and component associations 1142
  host header 1137
  installing to the first available new site number 1134
  INSTALLSHIELD_SSI_PROP 1128
  legacy object support 1149
  nested virtual directory 1133
  run-time requirements 1127
  scanning a Web site and importing its settings into InstallShield 1130
  site number 1134
  SSIEnableCmdDirective 1128
  SSL certificate 1138
  supported versions 1125
  TCP port number 1134
  timeout parameters 1148
  uninstalling application pools 1144
  uninstalling virtual directories 1143
  uninstalling Web service extensions 1144
  uninstalling Web sites 1143
  using InstallScript text substitution for 1154
  using Windows Installer properties for 1152
  virtual directory creation 1129
  Web service extension 1141
  Web site creation 1129
IIS Metabase and IIS 6 Configuration Compatibility feature
  detecting the presence of 1149
IIS_VERSION property 1126
IISscan.exe 3229
Import DIM Wizard 2021
Importing
  Visual Studio project into InstallShield project 404
Importing dialogs 874
ImportINIFile method 2895
ImportProject method 2864
ImportRegFile method 2896
ImportStrings method 2865
INI File Changes view 2285
InsertCustomAction method 3038
Installation from Another InstallScript Installation 794
Installation Information view 2111
Installation projects 258
  changing name or location 271
  creating 268
  opening 269
  saving 271
  using one project to create setups for multiple product configurations 1212
  using the automation interface 2822
  working with 257
Installation sequence 839
Installations 254, 459, 501
  creating 459
  definition 254
  organizing 501
INSTALLDIR 481
  vs. TARGETDIR 468
Installing Windows Installer 633
InstallScript 751
  function call tips 309
  keyboard shortcuts for coding 316
InstallScript engine 752
  verifying scripting run time 3209
InstallScript engine as UI handler for InstallScript MSI projects 488
InstallScript installation projects 263
InstallScript MSI installation projects 264
  InstallScript engine as external vs. embedded UI handler 488
InstallScript MSI Object project 383
InstallScript User Interface Type setting 494
InstallScript view 2364
InstallScript vs. Basic MSI installation projects 258
InstallShield Best Practice Suite 1303
InstallShield Cabinet and Log File Viewer 1677
InstallShield MSI Diff 1671
InstallShield MSI Grep 1676
InstallShield MSI Query 1674
InstallShield MSI Sleuth 1675
InstallShield Prerequisite Editor 2557
InstallShield prerequisites 616
  64-bit support 1435
  adding to InstallScript projects 614

adding to redistributables gallery 609
adding to Windows Installer-based projects 612
admnistrative privileges required 1444
allowing end users to skip 1445
associating with features 619
behavior for 1444
command-line parameters for 1441
configuring a release that includes 621
creating and modifying 1433
dependencies for 1448
files for 1438
hiding from the installation list 1445
installation conditions for 1435
installation order for 620
MySQL Connector ODBC 638
Oracle 11g Instant Client 639
overview 616
planning for potential issues 1448
properties for 1434
removing from features 620
removing from projects 615
removing from redistributables gallery 609
restarting target machine after prerequisite installation 1443
running installations with 625
saving 1449
setting a run-time location for an individual one 623
setting release flags and locations 624
setting the build-time location for 622
showing progress bar for 1447
specifying an alternate download URL 1434
specifying machine reboot 1448
specifying parameters for installing 1440
specifying the required execution level for 1199
specifying the run-time location for 1203
uninstalling applications with 629
URLs for downloading 1439
InstallShield tables 2530
ISAlias 2531
ISCOMCatalogAttribute 2532
ISCOMCatalogCollection 2533
ISCOMCatalogCollectionObject 2533
ISCOMCatalogObject 2534
ISCOMPlusApplication 2534
ISCustomActionReference 2535
ISIISItem 2536
ISIISProperty 2536
ISProductConfigurationInstance 2538, 2539
InstallShield UWP App Suitability Suite 1293
InstallShield Virtualization Suitability Suite 1278
INSTALLSHIELD_IIS_NEXT_NEW_SITE_NUMBER 1134
INSTALLSHIELD_SSI_PROP property 1128
InstanceId property 1581
Internationalization 1565
Internet distribution 1338
proxy server support 1338
Internet Explorer prerequisite 616
Internet Information Services 1125
application creation 1129
application mappings 1148
appllication pool 1140
ASP.NET version 1145
configuring error messages for 1151
Enable32BitAppOnWin64 1146
error -2147467259 1125
feature and component associations 1142
host header 1137
installing to the first available new site number 1134
INSTALLSHIELD_SSI_PROP 1128
lASP.NET platform 1146
legacy object support 1149
nested virtual directory 1133
run-time requirements 1127
scanning a Web site and importing its settings into InstallShield 1130
site number 1134
SSIEnableCmdDirective 1128
SSL certificate 1138
supported versions 1125
TCP port number 1134
timeout parameters 1148
uninstalling application pools 1144
uninstalling virtual directories 1143
uninstalling Web service extensions 1144
uninstalling Web sites 1143
virtual directory creation 1129
Web site creation 1129
Invoker 1199
IS_BROWSE_FILEBROWSED 928
IS_IIS_DO_NOT_USE_REG 1152
IS_PRINT_DIALOG property 933
IS_PS_EXECUTIONPOLICY 820
IS_SCRIPT_TAG 397
IS_SQLSERVER_ALIAS_ONLY 1096
IS_SQLSERVER_AUTHENTICATION 1084
IS_SQLSERVER_CONNECTIONS_TO_VALIDATE 1095
IS_SQLSERVER_CXNS_ABSENT_FROM_INSTALL 1095
IS_SQLSERVER_DATABASE 1084
IS_SQLSERVER_DO_NOT_USE_REG 1095
IS_SQLSERVER_LOCAL_ONLY 1095
IS_SQLSERVER_PASSWORD 1084
IS_SQLSERVER_REMOTE_ONLY 1096
IS_SQLSERVER_SERVER 1084
IS_SQLSERVER_USERNAME 1085
IS_VM_DETECTED 1598
IS_VM_TYPE 1598
ISAlias table 2531
ISBP01 1305

ISBP02 1306
ISBP03 1306
ISBP04 1307
ISBP05 1308
ISBP06 1308
ISBP07 1309
ISBP08 1310
ISBP09 1311
ISBP10 1311
ISBP11 1312
ISBP12 1313
ISBP13 1313
ISBP14 1314
ISBP15 1315
ISBP16 1316
ISBP17 1316
ISBP18 1317
ISBP19 1317
ISBP20 1318
ISBPs 1303
ISBuild.exe 1178
    command-line parameters 3228
ISChainPackageCommit custom action 3201
ISChainPackagePrepare custom action 3202
ISChainPackageRollback custom action 3202
ISClrWrap table 818
ISCmdBld.exe 1173
    command-line parameters 3220
    syntax 3220
ISCOMCatalogAttribute table 2532
ISCOMCatalogCollection table 2533
ISCOMCatalogCollectionObject table 2533
ISCOMCatalogObject table 2534
ISCOMPlusApplication table 2534
ISComponentServiceCosting custom action 3202
ISComponentServiceFinalize custom action 3202
ISComponentServiceInstall custom action 3202
ISComponentServiceRollback custom action 3202
ISComponentServiceUninstall custom action 3202
ISCustomActionReference table 2535
ISDEBUGLOG 1688
ISDetectVM 1598
ISICE01 1259
ISICE02 1260
ISICE03 1260
ISICE04 1261
ISICE05 1261
ISICE06 1262
ISICE07 1263
ISICE08 1264
ISICE09 1265
ISICE10 1265
ISICE11 1266
ISICE12 1268
ISICE16 1268
ISICE17 1269
ISICE18 1270
ISICE19 1270
ISICE20 1273
ISICE21 1274
ISICE22 1274
ISICE23 1275
ISICE24 1276
ISICE25 1276
ISICE26 1277
ISICEs 1256
iSign.exe 1208
    command-line parameters 3230
ISIIS6APPPOOLSUPPORTS32BIT 1146
ISIISCleanup custom action 3202
ISIISCosting custom action 3202
ISIISInstall custom action 3203
ISIISItem table 2536
ISIISMETABASECOMPATPRESENT 1149
ISIISProperty table 2536
ISIISRollback custom action 3203
ISIISUninstall custom action 3203
ISInstallPrerequisites 3203
ISJITCompileActionAtInstall custom action 3203
ISJITCompileActionAtUnInstall custom action 3203
ISLockPermissionsCost custom action 3203
ISLockPermissionsInstall custom action 3203
ISNetApiInstall custom action 3203
ISNetApiLogonGroup 745
ISNetApiLogonPassword 745
ISNetApiLogonUsername 745
ISNetApiRollback custom action 3204
ISNetCreateIniForOneUser custom action 3204
ISNetDeleteIniFile custom action 3204
ISNetGetGroups custom action 3204
ISNetGetServers custom action 3204
ISNetGetUsers custom action 3204
ISNetSetLogonName custom action 3204
ISNetValidateLogonName custom action 3204
ISNetValidateNewUserInformation custom action 3204
ISNetworkSharesCosting custom action 3205
ISNetworkSharesFinalize custom action 3205
ISNetworkSharesInstall custom action 3205
ISNetworkSharesRollback custom action 3205
ISNetworkSharesUninstall custom action 3205
ISO/IEC 19770-2 498
IsPlaying method 1219
ISPrint custom action 933, 3205
ISProductConfigName variable 1181
ISProductConfigurationInstance table 2538, 2539
ISQuickPatchFinalize custom action 3205
ISQuickPatchFixShortcut custom action 3205
ISQuickPatchHelper custom action 3206

ISQuickPatchInit custom action 3206
ISQuickPatchInit9X custom action 3206
ISQuickPatchInit9X2 custom action 3206
ISReleaseFlags property 807
ISReleaseName variable 1181
ISReleasePath variable 1181
ISReleaseUsesShallowFolderPaths variable 1181
ISRunSetupTypeAddLocalEvent custom action 3206
ISSCRIPT_ENGINE_VERSION property 1687
ISSCRIPT_VERSION_MISSING property 1687
ISSCRIPT_VERSION_OLD property 1687
ISSearchReplaceCosting custom action 3206
ISSearchReplaceFinalize custom action 3206
ISSearchReplaceInstall custom action 3206
ISSearchReplaceRollback custom action 3207
ISSearchReplaceUninstall custom action 3207
ISSelfReg 652
ISSelfReg table 652
ISSelfRegisterCosting action 652
ISSelfRegisterCosting custom action 3207
ISSelfRegisterFiles action 652
ISSelfRegisterFiles custom action 3207
ISSelfRegisterFinalize action 652
ISSelfRegisterFinalize custom action 3207
ISSetAllUsers 864
ISSetAllUsers custom action 3207
ISSetTARGETDIR custom action 3207
ISSetup.dll 752
ISSetupFilesCleanup custom action 3207
ISSetupFilesExtract custom action 3207
ISSQLQueryDatabases custom action 3208
ISSQLServerCosting custom action 3208
ISSQLServerFilteredList custom action 3208
ISSQLServerFinalize custom action 3208
ISSQLServerInitialize custom action 3208
ISSQLServerInstall custom action 3208
ISSQLServerList custom action 3208
ISSQLServerRemoveLoginInfo custom action 3208
ISSQLServerRollback custom action 3208
ISSQLServerRollbackLoginInfo custom action 3209
ISSQLServerUninstall custom action 3209
ISSQLServerValidate custom action 3209
ISSQLServerWriteLoginInfo custom action 3209
ISSQLSRV.dll 1086
ISSupportPerUser 747
ISUnSelfRegisterFiles action 652
ISUnSelfRegisterFiles custom action 3209
ISUWP01 1295
ISUWP02 1296
ISUWP03 1296
ISUWP04 1296
ISUWP05 1297
ISUWP06 1297
ISUWP07 1297
ISUWP08 1298
ISUWP09 1298
ISUWP10 1299
ISUWP11 1299
ISUWP12 1300
ISUWP13 1300
ISUWP14 1301
ISUWP15 1301
ISUWP16 1301
ISUWP17 1302
ISUWP18 1302
ISUWP19 1302
ISUWP20 1303
ISVerifyScriptingRuntime custom action 3209
ISVICE01 1280
ISVICE02 1281
ISVICE03 1282
ISVICE04 1282
ISVICE05 1283
ISVICE06 1284
ISVICE07 1284
ISVICE08 1285
ISVICE09 1286
ISVICE10 1286
ISVICE11 1287
ISVICE12 1288
ISVICE13 1289
ISVICE14 1289
ISVICE15 1290
ISVICE16 1291
ISVICE17 1292
ISVICE18 1292
ISVICE19 1293
ISVICEs 1278
ISWiAdvancedFile object 2869
ISWiAdvancedFiles collection 3016
IswiAuto 2822
ISWiComponent object 2872
ISWiComponents collection 3018
ISWiComponentSubFolder object 2901
ISWiComponentSubFolders collection 3020
ISWiCustomAction object 2871
ISWiCustomActions collection 3022
ISWiDynamicFileLinking object 2904
ISWiDynamicFileLinkings collection 3023
ISWiEnvironmentVar object 2906
ISWiEnvironmentVars collection 3024
ISWiFeature object 2908
ISWiFeatures collection 3025
ISWiFile object 2919
ISWiFiles collection 3026
ISWiFolder object 2925
ISWiFolders collection 3027
ISWiLanguage object 2929, 3028

ISWiLanguage object (Advanced UI, Suite/Advanced UI) 3083
ISWiLanguages collection (Advanced UI, Suite/Advanced UI) 3176
ISWiObject object 2931
ISWiObjects collection 3029
ISWiPathVariable object 2932
ISWiPathVariable object (Advanced UI, Suite/Advanced UI) 3085
ISWiPathVariables collection (Advanced UI, Suite/Advanced UI) 3177
ISWiPathVariables object 3030
ISWiProductConfig object 2935
ISWiProductConfigs collection 3031
ISWiProject object 2822
AddProperty method 2844
GenerateGUID method 2864
ISWiProject object (Advanced UI, Suite/Advanced UI) 3053
ISWiProperties collection 3033
ISWiProperties collection (Advanced UI, Suite/Advanced UI) 3178
ISWiProperty object 2940
ISWiProperty object (Advanced UI, Suite/Advanced UI) 3089
ISWiRelease object 2941
CubFile property 2984
ISWiReleases collection 3034
ISWiRemoveFileObject object 2985
ISWiRemoveFiles collection 3035
ISWiRemoveFiles method 2897
ISWiSequence collection 3036
ISWiSequenceRecord object 2986
ISWiSetupFile object 2987
ISWiSetupFile object (Advanced UI, Suite/Advanced UI) 3090
ISWiSetupFiles collection 3039
ISWiSetupFiles collection (Advanced UI, Suite/Advanced UI) 3179
ISWiSetupType object 2988
ISWiSetupTypes collection 3040
ISWiShellProperties collection 3041
ISWiShellProperty object 2989
ISWiShortcut object 2990
ISWiShortcuts collection 3042
ISWiSISProperties collection 3043
ISWiSISProperty object 2997
ISWiStringEntries collection 3050
ISWiStringEntry object 3014
ISWiSuiteAction object (Advanced UI, Suite/Advanced UI) 3091
ISWiSuiteActionRef object (Advanced UI, Suite/Advanced UI) 3098
ISWiSuiteActionRefs collection (Advanced UI, Suite/Advanced UI) 3179
ISWiSuiteActions collection (Advanced UI, Suite/Advanced UI) 3180
ISWiSuiteCondition object (Advanced UI, Suite/Advanced UI) 3100
ISWiSuiteConditionParameter object (Advanced UI, Suite/Advanced UI) 3111
ISWiSuiteConditionParameters collection (Advanced UI, Suite/Advanced UI) 3180
ISWiSuiteConditions collection (Advanced UI, Suite/Advanced UI) 3181
ISWiSuiteDFLFilter object (Advanced UI, Suite/Advanced UI) 3112
ISWiSuiteDFLFilters collection (Advanced UI, Suite/Advanced UI) 3182
ISWiSuiteDynamicFileLink object (Advanced UI, Suite/Advanced UI) 3115
ISWiSuiteDynamicFileLinks collection (Advanced UI, Suite/Advanced UI) 3183
ISWiSuiteEvent method (Advanced UI, Suite/Advanced UI) 3079
ISWiSuiteEvent object (Advanced UI, Suite/Advanced UI) 3118
ISWiSuiteEvents collection (Advanced UI, Suite/Advanced UI) 3183
ISWiSuiteExitCondition object (Advanced UI, Suite/Advanced UI) 3120
ISWiSuiteExitConditions collection (Advanced UI, Suite/Advanced UI) 3184
ISWiSuiteFeature object (Advanced UI, Suite/Advanced UI) 3121
ISWiSuiteFeatures collection (Advanced UI, Suite/Advanced UI) 3185
ISWiSuiteFile object (Advanced UI, Suite/Advanced UI) 3128
ISWiSuiteFolder object (Advanced UI, Suite/Advanced UI) 3129
ISWiSuiteFolders collection (Advanced UI, Suite/Advanced UI) 3187
ISWiSuiteOperation object (Advanced UI, Suite/Advanced UI) 3134
ISWiSuitePackage object (Advanced UI, Suite/Advanced UI) 3140
ISWiSuitePackages collection (Advanced UI, Suite/Advanced UI) 3188
ISWiSuiteRelease object (Advanced UI, Suite/Advanced UI) 3163
ISWiSuiteReleases collection (Advanced UI, Suite/Advanced UI) 3188
ISXmlAppSearch custom action 3209
ISXmlCosting custom action 3210
ISXmlFinalize custom action 3210
ISXmlInstall custom action 3210
ISXmlRollback custom action 3210
ISXmlUnInstall custom action 3210
Itanium
and .NET Framework 634, 637

J

Java(TM) 2 Runtime Environment prerequisite 616
Jet 4.0 prerequisite 616
JRE prerequisite 616

K

Key files 555
Key paths 694

L

Language identifiers 1576
Language IDs for InstallScript 795
Language support 408
Language-dependent components, installing 1570
Last Disk folder, adding files to 868
Launch an Executable 822
LaunchCondition table 283
Launching
    InstallShield on 32-bit vs. 64-bit systems 247
    InstallShield with or without administrative privileges 246
LaunchProgramFileFromSetupCompleteSuccess custom action 3210
LaunchReadmeFileFromSetupCompleteSuccesscustom action 3210
Library file, creating 758
license agreement
    requiring end users to read 931
Lifecycle of an application 255
Line control 973
List box control 975
list box control, Suite/Advanced UI or Advanced UI project 1038
    configuring 1038
List view control 979
Live redistributables gallery 605
    downloading latest to your machine 607
    icons associated with 605
Local repository 276
    overview 276
LocalDB 1088
Localization 1049, 1565
Localizing the user interface 1049
Locked-down environment 1525
    patches for 1525
Log file
    debugging InstallScript actions in a Suite/Advanced UI installation 3253
    generate for a package in a Suite 3254
    generate for Basic MSI and InstallScript MSI Setup.exe 3247
    generate for Suite/Advanced UI or Advanced UI Setup.exe 3254
    including action information in 834
lower 31-bit 792

M

Maintenance 1157
    preparing installations for maintenance and uninstallation 1157
Major Upgrade 1494
Managed assembly 815
    custom action 815
        32 bit vs. 64 bit 818
Manifest 652
    sample 654
mapped-drive locations
    referencing in projects 246
Masked edit control 984
MDAC 2.8 prerequisite 616
Media view 2417
Merge Module projects 266
    adding dependencies 1453
    adding exclusions 1454
    selecting language 1452
    setting default destination folder 1452
Merge modules 2247
    about 2247
    adding in direct edit mode 1724
    adding to InstallScript projects 614
    adding to redistributables gallery 611
    adding to Windows Installer-based projects 612
    browsing for 610
    downloading to your computer 607
    Live redistributables gallery 605
    obtaining updates for 408
    removing from projects 615
    removing from the redistributables gallery 611
    setting the build-time location for 630
Microsoft .NET Framework 635
    version 2.0, 64-bit 634, 637
    version 3.0 637
    version 3.0, 64-bit 634, 637
Microsoft .NET Object Wizard for InstallScript 2026
Microsoft Visual Studio, integration with InstallShield 1728
Microsoft Windows Azure SQL support 1107
Microsoft XML Core Services 712
Migrating from InstallShield - Windows Installer Edition 398
Migrating from InstallShield Professional 384
Minimum CSD Version setting for an InstallShield prerequisite 1898
Minor Upgrade 1495
mode conditions 1601
Modifying

InstallShield prerequisites 1433
response transform 1345
transform 1343
Move method (Advanced UI, Suite/Advanced UI) 3110
MoveDown method (Advanced UI, Suite/Advanced UI) 3114, 3162, 3163
MoveTo method (Advanced UI, Suite/Advanced UI) 3099
MoveUp method (Advanced UI, Suite/Advanced UI) 3115
MSBuild 1189
MSDE 2000 prerequisite 616
MSI database in Direct Edit Mode 270
MSI Debugger view 2523
MSI Diff 1671
MSI Diffs 1667
MSI Grep 1676
MSI Query 1674
MSI redistributables 633
MSI Sleuth 1675
msidbComponentAttributesShared 572
MsiExec.exe 3233
MsiPatchOldAssemblyFile 1558
MsiPatchOldAssemblyName 1558
MsiRMFilesInUse dialog 935
MST Wizard 2034
MSXML 712
Multiline registry value 696
Multiple Package Shared Component setting 572
Multiple Product Configurations, creating with one project 1212
Multiple-instance support 1579
configuring 1580
configuring ProductCode for 1582
configuring properties for 1583
naming an instance 1581
patches 1587
run-time behavior 1587
run-time requirements for 1579
Multiple-package installations 1163
overview 1163
using transaction processing 1163

N

Namespaces 727
adding prefixes to elements for 728, 729, 730
declaring in an XML file 728, 731
removing prefixes from elements 730
Nested installations 826
Network repository 276
overview 276
setting up 276
Never Overwrite component property 565
New Project dialog 1853
Non-administrator patch 1525
requirements 1525
notifying end users about updates 1077
adding a dialog to check for updates 1080
creating a shortcut to check for updates 1079
disabling in a project 1078
FlexNet Connect 2150
installation files needed for 1079
registering your application for 1081

O

object expressions in Advanced UI and Suite/Advanced UI
installations 3191
Feature object 3191
File object 3193
Parcel object 3195
Platform object 3197
Registry object 3199
Objects 2247, 2252
about 2252
adding to InstallScript projects 614
adding to Windows Installer-based projects 612
building 1477
creating 1457
designing 1458
distributing 1479
downloading to your computer 607
files for 1460
internationalizing 1463
Live redistributables gallery 605
localizing 1463
managing gallery of 605
methods 1471
object status properties 1472
Objects view 2252
object-specific functions 1481
obtaining updates for 408
properties 1465
registering 612
removing from projects 615
script-defined destinations 1483
Status properties 1472
system variables 1482
testing 1478
unsupported constants 1480
unsupported features 1476
unsupported functions 1480
upgrading InstallScript projects that have 381
wizards 1461
ODBC Resources view 2282
OnBegin
in a Suite/Advanced UI installation 1398
One-Click Install installations 1213
authenticating users 1226

creating 1223
debugging 1226
Ether object 1217
InstallShield versus InstallShield Professional 1214
passing data to the launched installation 1225
Player object 1215
setting language 1225
silent 1225
system requirements 1223
upgrading from InstallShield 12 or earlier 363
OnEnd
in a Suite/Advanced UI installation 1398
OnPackagesConfigured 1398
OnPackagesConfiguring 1398
OnRebooting
in a Suite/Advanced UI installation 1398
OnResuming
in a Suite/Advanced UI installation 1398
OnStaged 1398
OnStaging 1398
Open method 1216
Open MSP Wizard 2033
Open Transform Wizard 2034
Opening
.msi file 270
installation projects 269
InstallShield Prerequisite Editor 2557
OpenProject method 2866
OpenProject method (Advanced UI, Suite/Advanced UI) 3081
operating system requirements 283
when installations check for 284
Oracle 11g Instant Client prerequisite 1108
Oracle support 1108
for 64-bit target systems 1108
Organization view 2150
Orphaned directory entries 1669
Other disk folder, adding files to 869
Other window styles 1871
Output window 1769
docking or undocking 305

P

Package Manager 1379
Packages view 2206
Palm OS device installations
requirements 239
Password protection 1202
for a patch 1530
for a QuickPatch 1539
Patch 1521
sequences for small updates 1521
shared component in 572
uninstallation of 1524
Patch Configuration 1496
Patch Design view 1496
Patching 2033
a product whose installation supports multiple instances 1587
assemblies in the global assembly cache 1558
vs. upgrading 1505
with non-administrator patches 1525
Path 986
variables 2419
converting standard paths 510
environment 507
predefined 504
registry 506
standard 505
path variables and DIMs 661
Path Variables view 2419
Per-user vs. per-machine installations
registry entries 697
pinning shortcuts 671
to the Windows taskbar or Start menu 673
Pkgmgr.exe 1379
Platform ID setting for an InstallShield prerequisite 1898
Play method 1219
PowerShell script 820
POWERSHELLVERSION 820
Predefined Search 1780
Prerequisite Editor 2557
Prerequisites 616
64-bit support 1435
adding to InstallScript projects 614
adding to redistributables gallery 609
adding to Windows Installer-based projects 612
administrative privileges required 1444
allowing end users to skip 1445
associating with features 619
behavior for 1444
command-line parameters for 1441
configuring a release that includes 621
creating and modifying 1433
dependencies for 1448
files for 1438
hiding from the installation list 1445
installation conditions for 1435
installation order for 620
MySQL Connector ODBC 638
obtaining updates for 408
Oracle 11g Instant Client 639
overview 616
planning for potential issues 1448
properties for 1434
removing from features 620
removing from projects 615
removing from redistributables gallery 609

restarting target machine after prerequisite installation 1443
running installations with 625
saving 1449
setting a run-time location for an individual one 623
setting release flags and locations 624
setting the build-time location for 622
showing progress bar for 1447
specifying an alternate download URL 1434
specifying machine reboot 1448
specifying parameters for installing 1440
specifying required execution level 1199
specifying the run-time location for 1203
uninstalling applications with 629
URLs for downloading 1439
User Account Control prompts and 462
Print button 933
Processor Architecture setting for an InstallShield prerequisite 1898
Product (OS) Type setting for an InstallShield prerequisite 1898
Product configurations 1167
using one project to create setups for multiple product configurations 1212
Product name
displayed on Setup.exe's Properties dialog box 1329
displayed on Update.exe's Properties dialog box 1519
Products
conditional installation 479
destination folder 481
hierarchy of features and components for 254
Template Summary 495
ProgIDs 575
Program Compatibility Assistant 174
progress bar (Basic MSI projects) 1059
displaying with or without billboards 1059
Progress bar control 989
Progress dialog 834
specifying messages about actions being launched 834
ProgressIncrement event for automation interface 1745
ProgressMax event for automation interface 1745
Project Assistant 279, 414
Project templates 276
creating 277
using as project types 277
Project types 259
converting from one to another 272
converting or importing Visual Studio project into InstallShield project 404
Property Manager 2390
localizing properties 1717, 1718
private properties 1685
public properties 1685
required properties 1685
restricted public properties 1685
using 1714
Proxy server settings 1338
Proxy support for server application 1123
conditionally installing COM+ application 1124
Publishing components 584
Push button control 992
PVK2PFX 1205

Q

QuickPatch
built-in InstallShield custom actions for 1535
how InstallShield creates 1535
streamlining creation of 1535
streamlining limitations 1535
QuickPatch projects 267
Create New QuickPatch Wizard 1968
custom action 2552
files 2552
history 2552
procedures for 1534
registry 2555

R

Radio button control 996
Radio button group control 998
Rebooting target machine 1443
during an installation or uninstallation (InstallScript or InstallScript MSI) 1319
specifying for prerequisites 1440
Redistributables 2247
.NET Framework and language packs 635
adding merge modules to gallery of 611
adding to InstallScript projects 614
adding to Windows Installer-based projects 612
downloading latest ones to your machine 607
InstallShield 603
InstallShield prerequisites 616
live gallery for 605
managing gallery of 605
obtaining updates for 408
Reevaluate condition 563
REG files 693
Multi-line string support 1850
registering COM servers 645
Registry 424, 445, 2281
enabling or disabling reflection of 571
Reg-Free COM 652
view 2281
Registry entries 680
creating keys 683
exporting REG files 693

finding text in a component's registry data 690
flags 688
for per-user installations 697
importing REG files 693
key paths 694
registering values 686
setting uninstall behavior 691
Registry reflection 571
Release flags 1208
Release Wizard 419, 439
Release wizard 2041
ReleasePackager.exe 3231
Releases
leaving files uncompressed in a disk image 1198
Remote installation 596
RemoveComponent method 2917
RemoveComponentSubFolder Method 2898
RemoveDynamicFileLinking method 2899
RemoveEnvironmentVar method 2899
RemoveFeature method 2868
RemoveFile method 878
RemoveMergeModule method 2918
RemoveObject Method 2919
RemovePackage method (Advanced UI, Suite/Advanced UI) 3127
RemoveRemoveFile method 2901
RemoveSequenceRecord method 3038
Removing
InstallShield prerequisites from projects 615
InstallShield prerequisites from redistributables gallery 609
merge modules and objects from projects 615
merge modules from redistributables gallery 611
server locations 1347
Repository 276
overview 276
setting up network 276
Required Execution Level setting 1199
requirements for target machines 283
when installations check for 284
requirements for target systems 239
Response transform 1345
creating 1345
modifying 1345
Restarting target machine 1443
affect on UAC prompts for Windows Vista and later 462
during an installation or uninstallation (InstallScript or InstallScript MSI) 1319
specifying for prerequisites 1440
Return values 2677
Setup.exe run time (Basic MSI and InstallScript MSI) 2677
Setup.exe run time (InstallScript) 2674
Setup.exe run time (Suite/Advanced UI and Advanced UI) 2681
Right-to-left language support 410
Roles 1379
enabling Windows features and 1379
Rollback 858
RTL language support 410
Running an installation 417, 1232
multiple times 487
running multiple .msi files simultaneously 616
with InstallShield prerequisites 625

S

Save As dialog box 1916
SaveProject method 2869
SaveProject method (Advanced UI, Suite/Advanced UI) 3082
Saving
InstallShield prerequisites 1449
project with new name or location 271
Script editor 2366
Script Files 754
Scrollable text control 1002
Sd Dialog Static Text Fields 891
secondary window 1019
adding a blank one 1031
adding a control to 1033
background for 1025
displaying ampersands in text 1034
keyboard shortcuts for 1034
tab order for 1034
Security 1205
Selection tree control 1004
Self-extracting executable files 1179
building 1197
building from the command line 1179
using ReleasePackager.exe to build 3231
Self-registering files 425, 446, 650
specifying order of self-registration 652
Self-registration 650
COM server support 646
Sequences 2367
Administration sequence 852
Advertisement sequence 850
custom actions 859
Execute sequence 854
for installing prerequisites 620
for small-update patches 1521
insert an action 855
Installation sequence 839
User Interface sequence 854
Server 1124
location
adding 1346
change the order of multiple 1348
modifying 1347

removing 1347
specify for application proxy 1123
Server Configuration view 2320
services 744
installing, starting, stopping, deleting, uninstalling, and configuring 744
setAllUsersProfile2K custom action 3210
SetAllUsersProfileNT custom action 3210
SetARPINSTALLLOCATION custom action 3210
SetARPReadme (custom action) 497
SetProperty Method 1220
Settings.xml 317
Setup Best Practices 541
Component wizard option 544
setup design monitoring 1855
Setup Design views 2153
Setup prerequisites
obtaining updates for 408
User Account Control prompts and 462
Setup types 418, 2204
Setup_UI.xml 1349
Setup.exe 1327
command-line parameters (Basic MSI, InstallScript, InstallScript MSI) 3238
command-line parameters (Suite/Advanced UI and Advanced UI) 3252
customizing properties for 1329
including InstallShield prerequisites in 625
InstallScript projects 3256
log, for Basic MSI 3247
log, for packages in Suites 3254
log, for Suite/Advanced UI and Advanced UI installations 3254
redistributing Windows Installer 633
return values (Basic MSI and InstallScript MSI) 2677
return values (InstallScript) 2674
return values (Suite/Advanced UI and Advanced UI) 2681
run-time errors (Basic MSI and InstallScript MSI) 2677
run-time errors (InstallScript) 2674
run-time errors (Suite/Advanced UI and Advanced UI) 2681
specifying the icon for 1334
Setup.ilg 1681
converting to .txt file 1683
opening 1682
overview 1681
viewing 1681
Setup.ini 1226
Setup.log 1249
Setup.rul 752
Setup.xml 1349
SETUPEXEDIR 1708
SetupIni.exe 3233
SetupUI.dll 1349
setUserProfileNT custom action 3211
Shared files 563
shield icon 462
Shortcuts 423, 444, 663
creating a shortcut to check for updates 1079
creating shortcuts 554
creating uninstallation shortcuts for Basic MSI projects 677
creating uninstallation shortcuts for InstallScript and InstallScript MSI projects 678
icons for 667
pinning 671
setting properties 666
view 2257
ShowMsiLog custom action 3211
sideloading 1364
an app package (.appx) 1364
Signing files, digitally 1205
Silent installations 1233
Silent uninstallation from the command line 1252
Single executable files, creating 1197
skins, for InstallScript and InstallScript MSI dialogs 895
limitations of 895
Small update 1521
patch sequences for 1521
software identification tag 498
and SCCM app model data 499
Source control 1725
command-line 1743
Source location 565
SOURCELIST property
adding values to 1346
changing order of values in 1348
editing values for 1347
removing values from 1347
SQL 2347
Adding a connection 1087
Adding SQL script 1089
auto completion 307
automatic indentation 313
Automation
AddSQLRequirement Method 3000
AddSQLScript Method 3001
AddSQLScriptError Method 3011
AddSQLScriptEx method 3001
AddSQLScriptReplacement Method 3012
AddSQLServerConnection Method 2846
DeleteSQLConnection Method 2860
DeleteSQLReplace Method 3012
DeleteSQLScript Method 3002
DeleteSQLScriptError Method 3013
GetDatabaseServer Method 3003
InsertSQLScript Method 3003
ISWiSQLConnection Object 2998

ISWiSQLConnections Collection 3044
ISWiSQLDatabaseServer Object 3004
ISWiSQLDatabaseServers Collection 3045
ISWiSQLReplace Object 3006
ISWiSQLReplaces Collection 3046
ISWiSQLRequirement Object 3007
ISWiSQLRequirements Collection 3047
ISWiSQLScript Object 3009
ISWiSQLScriptError Object 3013
ISWiSQLScriptErrors Collection 3048
ISWiSQLScripts Collection 3049
RemoveSQLRequirement Method 3004
SetPredefinedRequirement Method 3008
bookmarks in script 306
connections and Windows Installer properties 1085
creating a SQL Server database through SQL script 1116
creating an Oracle schema through SQL script 1118
editing a SQL script 1092
Importing Databases 1996
keyboard shortcuts for entering 316
line numbers, showing or hiding 312
LocalDB 1088
Oracle support for 64-bit target systems 1108
overriding default run-time behavior 1095
overriding the default protocol 1088
requirements for InstallScript and InstallScript MSI projects 1086
requiring a SQL Server-side installation 1104
resolving issues with Unicode and Oracle databases 1115
saving Windows Installer properties for 1084
setting up a database server type condition for SQL scripts 1097
specifying launch order for SQL scripts 1094
syntax highlighting 310
using InstallScript text substitution to modify scripts for 1102
using Windows Installer properties to modify scripts for 1100
Windows Azure support 1107
working with the script editor for 306
SQL Server Express Edition prerequisite 616
SQL Server Native Client prerequisite 1107
SQLCONV.obl 1086
SQLRT.dll 1086
SQLRT.ini 1086
SQLRT.obl 1086
SSL certificate 1138
Standalone Build 1184
STANDARD_USE_SETUPEXE property 1687
StatusMessage event for automation interface 1745
streamlining QuickPatch packages 1535
String entries
accessing in InstallShield 1051
automating export and import of for translations 1740
editing 1054
for multiple languages 1050
using in InstallScript code 757
view 2415
String table entries, creating 453
Strings
automating export and import of, for translations 1740
style 1021
applying a text style to text in the wizard interface 1023
defining a custom one for the wizard interface 1022
for the wizard interface 1021
selecting the wizard format 1029
Subscriptions 905
subweb
adding files to 1138
subweb, configuring error messages for 1151
Suite/Advanced UI installation 261
about InstallScript packages in 1359
Add or Remove Programs entry for 1402
adding .appx packages to 1364
adding .exe packages to 1362
adding .msi, .msp, and transaction packages to 1356
adding an InstallScript package to 1358
adding packages from InstallShield projects to 1361
adding Web Deploy packages to 1363
automatic updates for 1418
custom package folder names 1378
dependency packages 1370
differences with Advanced UI installations 1351
downloadable updates for 1418
dynamic file linking 1365
feature state 1408
feature-package associations 1373
files included in 1349
formatted expressions 1410
guidelines for adding packages to 1355
importing .prq files into 1369
importing InstallShield prerequistes into 1369
installation order of packages 1371
InstallScript actions in 1390
InstallScript events, functions, and variables for 1358, 1359
logging a 1423
logging packages in a 1427
managed-code actions in 1391
mode condition 1401
object expressions 1412, 3191
package operations for 1376
passing command-line parameters to packages in a 1403
password protecting 1422
primary packages 1370
properties 1709
requirements for 239
run-time locations of packages 1375
Setup.exe command-line parameters 3252

shared packages 1373
system searches in 1412
troubleshooting issues with 1423
UAC prompts 1413
unexpected maintenance mode 1423
Support files 865
supporting 283
supressing shortcuts
from the Windows Start screen 671
Syntax
highlighting 310
syntax folding 314
Sysnative folder 514
System Configuration view 2253
System Restore 769
System Search view 2389
about registry searches 1653
System Search Wizard 2082
about registry searches 1653
System.AppUserModel.ExcludeFromShowInNewInstall 675
System.AppUserModel.PreventPinning 673

T

tagging 498
to identify a product 498
to include SCCM app model data 499
Target languages, selecting 451
target system requirements 239
TARGETDIR 468
vs. INSTALLDIR 468
Team Explorer 1735
Team Foundation Server, integration with 1735
templates 276
for projects 276
terminate process 819
Test running a setup 1232
Text area control 1008
TFS, integration with 1735
theme
defining a theme for controls in the wizard interface 1027
Theme, dialog 912, 927
Tile configurations
configurating desktop app tile on Start screen 679
settings 2278
timestamp server for digital signatures 318
Transaction processing 1163
overview 1163
Transform wizard 2093
Transforms 1341
applying 1343
creating by comparing two .msi files 1342
creating by starting with a single .msi package 1342
customizing default responses for 1345
editing 268
Transitive components 563
Translated string entries 408
Troubleshooting 2661

U

UAC and installations 462
Unicode 409
uninstallation shortcut
creating for a Basic MSI project 677
creating for an InstallScript or InstallScript MSI project 678
Uninstalling 629
applications with prerequisites 629
executing script code only during uninstallation 1158
InstallScript functions that are logged for uninstallation 1159
patches 1524
preparing installations for uninstallation and maintenance 1157
Unsupported functions 780
Update.exe 3238
customizing properties for 1519
specifying the icon for 1520
Updates 408
for InstallShield 408
Updating 1559
automatically notifying end users about 1077
InstallShield 408
Upgrade warnings 2712
Upgrades view 2424
Upgrading 1559
automatically notifying end users about 1077
InstallScript MSI Object project 383
preventing downgrades while 468
projects 2424
vs. patching 1505
upper 31-bit 792
User Account (Adding or Setting) 872
User Account Control (UAC) and installations 462
User Interface sequences 854
User Interface view 2401
Users' default selections for dialog boxes 1345
UWP app support 1293
creating UWP app packages 1231
deploying UWP app packages 1339
InstallShield UWP App Suitability Suite 1293

V

Validation 1252
ICEs 1256
InstallShield UWP App Suitability Suite 1293

ISICEs 1256
ISUWPs 1293
ISVICEs 1278
of custom actions 806
performing on demand 1255
specifying when to perform 1254
specifying which ICEs to run 1254
upgrading and patching 1542
viewing results 1255
ValidationFiles folder 1255
Variables
using bit flags in 771
VB .NET 1728
Version numbers, in InstallScript setups 477
View Filter 681
View List 2111
Virtual directories 1129
virtual directory
adding files to 1138
configuring error messages for 1151
Virtual machines 1598
detecting the presence of 1598
Virtual PC 1598
virtualization
specifying the build output location 325
Visual Studio .NET Wizard for Visual Basic .NET and Visual C# .NET 2104
Visual Studio Team Foundation Server
integrating with InstallShield 1735
VMConfigurations.xml 326
VMware 1598
voicewarmupx 1838
Volume cost list control 1012
Volume select combo control 1015
VSSolutionFolder 1729

W

WatchScroll custom action 931
Web Deploy support 1363
Web installations 2048
Web Service Extensions 1141
Web services 1735
deploying on a target machine 1155
Web site
adding files to 1138
configuring error messages for 1151
Windows API Function 789
Windows Azure SQL support 1107
Windows Installer engine requirements 239
Windows Installer properties 1685
and deferred, commit, or rollback custom actions 809
associating with string entries 1717, 1718
complete list 1687
public, private, restricted, and required 1685
using the Property Manager 2390
Windows Installer redistributables 633
version 4.5 634
Windows Installer tables 2530
Windows logo program 461
Windows Management Instrumentation prerequisite 616
Windows Mobile device installations
requirements 239
Windows roles and features 1379
enabling 1379
Windows Server Core 1168
Windows services 744
installing, starting, stopping, deleting, uninstalling, and configuring 744
WiseScriptFinalize custom action 3211
WiseScriptInitialize custom action 3211
WiseScriptRollback custom action 3211
wizard page 1019
actions for a control on 1042
adding a control to 1033
adding a predefined one 1030
adding a SQLLogin predefined wizard page 1120
background for 1025
configuring the order 1032
creating a new blank one 1030
displaying ampersands in text 1034
editing 1032
keyboard shortcuts for 1034
navigation buttons on 1036
scheduling actions on open or close 1042
tab order for 1034
validation for a control on 1040
WMI prerequisite 616
WOW64 1168

X

XML 2291
associating with a feature 715
Import XML Settings Wizard 2024
searching for XML data 1655
testing installation changes 731
testing uninstallation changes 733
using <, >, &, ', and " as content in elements 726
using InstallScript text substitution in 725
using namespaces in 727
using Windows Installer properties in 719, 724
XML (Saving an .ism in XML or Binary Format) 475